Technická univerzita v Liberci

FAKULTA PŘÍRODOVĚDNĚ-HUMANITNÍ a PEDAGOGICKÁ

Katedra: Katedra aplikované matematiky

Studijní program: B1101 Matematika

Studijní obor: Matematika – Informatika

INTERAKTIVNÍ VÝUKOVÝ MATERIÁL PRO ZÁKLADY PROGRAMOVÁNÍ

INTERACTIVE TEACHING MATERIALS FOR BASIC PROGRAMMING

Bakalářská práce: 13–FP–KAPi–001

Autor:

Jan BRZOBOHATÝ

Podpis:

Vedoucí práce: Ing. Jindra Drábková, Ph.D.

Počet

stran	grafů	obrázků	tabulek	pramenů	příloh
58	0	45	0	23	0

V Liberci dne: 18. 4. 2013

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta přírodovědně-humanitní a pedagogická
Akademický rok: 2011/2012

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan Brzobohatý**
Osobní číslo:	**P10000553**
Studijní program:	**B1101 Matematika**
Studijní obory:	**Informatika se zaměřením na vzdělávání**
	Matematika se zaměřením na vzdělávání
Název tématu:	**Interaktivní výukový materiál pro základy programování**
Zadávající katedra:	**Katedra aplikované matematiky**

Zásady pro vypracování:

Cílem práce je vytvořit interaktivní materiál pro výuku základů programování. V teoretické části (min. 15 stran) student popíše přístupy, které se v současné době používají při výuce programování. Součástí teoretické části práce bude porovnání a zhodnocení jednotlivých přístupů. V praktické části student vytvoří aplikaci pro výuku základů programování. Aplikace by měla sloužit pro úplné začátečníky a měla by obsahovat jak teoretické základy tak praktická cvičení.

Rozsah grafických prací: **dle potřeby**
Rozsah pracovní zprávy: **cca 45 stran**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

- **Pšenčíková, J.** *Algoritmizace*. **2. vydání. Kralice na Hané: Computer Media, 2009. ISBN 978-80-7402-034-6.**
- **Satrapa, P.** *Pascal pro zelenáče*. **Praha: Neokortex, 2000. ISBN 80-86330-03-6.**
- *Algorithms and Data Structures* **[online]. Dostupné z: http://www.algolist.net/.**

Vedoucí bakalářské práce: **Ing. Jindra Drábková, Ph.D.**
Katedra aplikované matematiky

Datum zadání bakalářské práce: **30. dubna 2012**
Termín odevzdání bakalářské práce: **26. dubna 2013**

L.S.

doc. RNDr. Miroslav Brzezina, CSc.
děkan

doc. RNDr. Miroslav Kouckỳ, CSc.
vedoucí katedry

V Liberci dne 30. dubna 2012

Čestné prohlášení

Název práce: Interaktivní výukový materiál pro základy programování

Jméno a příjmení autora: Jan Brzobohatý

Osobní číslo: P10000553

Byl/a jsem seznámen/a s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména § 60 – školní dílo.

Prohlašuji, že má bakalářská práce je ve smyslu autorského zákona výhradně mým autorským dílem.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval/a samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Prohlašuji, že jsem do informačního systému STAG vložil/a elektronickou verzi mé bakalářské práce, která je identická s tištěnou verzí předkládanou k obhajobě a uvedl/a jsem všechny systémem požadované informace pravdivě.

V Liberci dne: 18. 4. 2013

Jan Brzobohatý

Poděkování

Děkuji všem, bez kterých by tato práce nemohla vzniknout. Především děkuji paní Ing. Jindře Drábkové, Ph.D. za vedení bakalářské práce a cenné rady při jejím vypracovávání. Dále bych chtěl poděkovat Bc. Nině Adlerové za podporu a především za jazykovou korekturu bakalářské práce. Dále bych rád poděkoval přátelům, kteří otestovali můj program, a to Martinu Zvárovi, Jaroslavu Jakouběmu, Jakubu Trskovi, Ondřeji Vraštilovi a Kristiánu Kollerovi.

Anotace

Bakalářská práce je zaměřená na výuku základů programování. V práci jsou nejprve rozebrány a zhodnoceny jednotlivé přístupy k výuce základů programování. Jsou zde porovnána paradigmata programování, která jsou používána v současné době při výuce základů programování. Dále jsou rozebrány programovací jazyky, které jsou v dnešní době nejpoužívanější při výuce základů programování. Je zde také poukázáno na problematiku výuky na základních školách a popis metod pro výuku programování již v relativně nízkém věku. V druhé části práce je popsán program, který je výstupem bakalářské práce a nachází se na přiloženém CD. Jedná se o formu interaktivního materiálu pro základy programování. Program je zde popsán jak z uživatelského, tak především z programátorského hlediska. Je uzpůsoben pro začátečníky a obsahuje jednak teoretické základy programování, jednak konkrétní praktická cvičení navazující na teorii.

Klíčová slova: paradigmata programování, Pascal, programovací jazyk, programování, vývojový diagram, výuka programování

Annotation

This bachelor's thesis focuses on teaching the basics of programming. It first discusses and evaluates different educational approaches of programming. There are compared paradigms of programming used for teaching the basics of programming. Then it covers those programming languages that are most often used for this purpose these days. At this point, the paper introduces the specificities of teaching this subject at primary schools and methods suitable at this relatively early age. The second part of the thesis is devoted to the description of its practical result: a program located on the enclosed CD, suitable for presenting the basics of programming in an interactive way. Although the program is also introduced from the perspective of the user, even more attention is paid to the point of view of the programmer. The software itself is designed for beginners, and covers both the theoretical foundations of programming, and specific practical exercises related to theory.

Key words: programming paradigms, Pascal, programming language, programming, flowchart, teaching programming

Obsah

Seznam Obrázků

Úvod

Ve své práci se zabývám výukou základů programování. V teoretické části této práce rozebírám přístupy, které se v dnešní době při výuce základů programování používají. Autoři knih pro základy programování jsou v přístupech velice různorodí, ať už se to týká vhodných paradigmat programování pro začátečníky, tak i výběru vhodného prvního programovacího jazyka. Nejdříve popisuji dvě základní paradigmata programování, ke kterým se autoři knih přiklánějí a kterými jsou strukturované a objektově orientované programování. Dále popisuji programovací jazyky, které jsou v současné době používány pro výuku základů programování a kde se již autoři mnohem více názorově rozcházejí. Do teoretické části jsem zahrnul i dva velice populární programovací jazyky pro děti, ve kterých se programuje tzv. grafickou metodou, přestože jsem si vědom, že se již nejedná o klasické programování.

Cílem bakalářské práce je vytvoření interaktivního materiálu pro výuku základů programování. Jejím výstupem je aplikace, která by měla sloužit pro úplné začátečníky a obsahuje jak teoretické základy, tak praktická cvičení. Může sloužit jednak jako interaktivní pomůcka pro výuku programování nebo jako samo-výukový materiál. Výuka v aplikaci probíhá v jazyce Pascal a obsahuje teoretický výklad programovacího jazyka Pascal a interaktivní příklady, na kterých je daná teorie vždy patřičně procvičena. Nejdůležitější částí programu, tedy to, proč program vzniknul, jsou právě ony interaktivní příklady. Příklad je prezentován ve formě zdrojového kódu a zároveň ve formě vývojového diagramu. Uživatel si může program krokovat příkaz po příkazu, přičemž se mu krokování zobrazuje jak na zdrojovém kódu, tak na vývojovém diagramu. Při každém kroku může uživatel sledovat změny stavu proměnných a změny na standardním výstupu. Co se týče teorie v programu, vždy je vysvětleno pouze to nejnutnější k praktickému programování a důraz je kladen především na příklady. Velká část programu je věnována větvení a cyklům, které považuji za základy programování. Program je zakončen nečíselnými datovými typy (Char, String, Boolean, Pole).

1 Paradigmata programování

Paradigma programování je jednoduše řečeno styl, kterým lze programovat. Některé jazyky podporují více paradigmat a jiné jsou přímo určeny pro jedno konkrétní paradigma. [7 s. 16]

V této kapitole budou popsány a porovnány dvě nejpopulárnější paradigmata programování jak pro začátečníky, tak pro využití v praxi. Těmi jsou procedurální strukturované paradigma a objektově orientované paradigma. Ostatní paradigmata nejsou tak hojně používána v praxi a už vůbec ne pro začátečníky, z toho důvodu je zde nebudu rozebírat.

1.1 Procedurální paradigma

Procedurální paradigma, známé také jako imperativní paradigma, popisuje algoritmus dané úlohy krok za krokem. Určuje tedy přesný postup, jak úlohu řešit. Tento přístup je každému člověku blízký, protože ho používáme i v běžném životě. Například kuchařské recepty nebo návody k sestavení výrobku jsou také napsány krok za krokem v závislosti na určitém stavu jídla či výrobku (v programování stavu proměnných). [1]

Procedurální programování se dá rozdělit ještě na specifičtější druhy paradigmat, těmito druhy jsou nestrukturované a strukturované paradigma.

1.1.1 Nestrukturované paradigma

Nestrukturované paradigma je lineární sekvence příkazů, ve které se používá příkaz „GO TO“ pro skoky v programu, buď ve starších verzích na očíslovaný řádek, nebo v novějších verzích na návěstí. [7 s. 16]

Zástupci tohoto paradigmatu jsou například jazyky COBOL a FORTRAN.

1.1.2 Strukturované paradigma

Strukturované paradigma vzniklo hlavně z důvodu nepraktičnosti skoku (GO TO), který velice komplikuje ladění programu, protože nedává žádnou informaci o vykonávání programu. [1]

Strukturované programování využívá omezený sortiment používaných konstrukcí. Jsou povoleny jen takové, které mají jediný vstup a jediný výstup, jako třeba podmíněný příkaz či cyklus. Klasickým příkladem, který nesplňuje tyto požadavky strukturovaného programování, je již zmíněný skok (GO TO). [2 s. 199]

Dalším důležitým znakem strukturovaného programování je to, že většina podprogramů rozdělí svůj úkol na části a pro jejich řešení povolá další podprogramy. Používá se zde metoda

návrhu tzv. shora-dolu, kde postupně celkový úkol rozdělujeme na menší úkoly pro jednotlivé podprogramy a pokud možno těmto podprogramům rozdělíme úkol na další dílčí úkoly pro další podprogramy. Takto se snažíme program rozdělit co nejjemněji a vznikne nám tím jistá hierarchie podprogramů. [2 s. 199, 200]

Při dodržení těchto zásad se dosáhne sice delšího zdrojového kódu a pomalejšího programu, ale vznikne velice přehledný a jednoduše pochopitelný kód, což výrazně napomáhá pozdějšímu ladění. [2 s. 200]

Strukturované programování může ušetřit čas a energii při psaní jednoduchých programů. Je ideální pro vývoj malých programů, neboť při vytváření takového programu pomocí objektově orientovaného přístupu by uživatel strávil spoustu času a energie navrhováním tříd. Malé programy se ve strukturovaném kódu snadno udržují, protože se nacházejí v jednom souboru a vejdou se mnohdy i na jednu stránku. Strukturované programy jsou snadno čitelné a srozumitelné, protože lze číst zdrojový kód tak, jak je napsán v souboru, tedy od shora dolu. Je zde zaručeno, že neexistují žádné skoky nebo odkazy na jiné kusy kódu v jiných souborech, které by přehlednost snížili. [8]

Typickými zástupci strukturovaného paradigmatu jsou jazyky Pascal, C, ADA atd.

1.2 Objektově orientované paradigma

Objektové programování je svým způsobem modelování reálného světa. Objekty reálného světa jsou promítnuty do objektů programu. Tyto objekty mají určité vlastnosti a mohou vykonávat určité činnosti. Také mohou interagovat mezi sebou a navzájem se ovlivňovat pomocí zasílání zpráv. Každý objekt je instancí nějaké třídy, která určuje společné vlastnosti a činnosti pro objekty vycházející z této třídy. Jednotlivé objekty vycházející z jedné třídy se od sebe mohou lišit pouze v některých vlastnostech. Činnosti, které mohou provádět, se obecně nazývají metody. [4]

Objektově orientované programování by mělo mít tři základní vlastnosti, kterými jsou zapouzdření, dědičnost a polymorfismus. [4]

Zapouzdření znamená, že objekt nemůže přistupovat k vlastnostem ostatních objektů a jediné, jak je možné interagovat s druhým objektem, je pomocí metod, které daný objekt poskytuje. [1]

Dědičnost umožňuje třídám dědit vlastnosti a metody od jiných tříd, což vede k pomyslnému vytvoření jakési hierarchie tříd. Každý potomek si však své zděděné vlastnosti a metody může upravit podle svého a navíc může přidat další vlastnosti a metody specifické pouze pro něj. [1]

Polymorfismus zaručuje, že každý objekt ze stejné třídy bude poskytovat stejné rozhraní (metody), ale každý z těchto objektů se bude chovat jinak podle jeho vlastností (proměnných). [1]

Na rozdíl od strukturovaného programování se zde používá při navrhování tzv. metoda zdola nahoru. Nejdříve se určí elementární prvky (objekty) programu a pomocí těchto prvků se složí postupně celý program. [4]

Objektově orientované programování je vhodné pro vytváření a udržování obrovských softwarových a webových projektů. V případě těchto velkých projektů je objektově orientovaný přístup mnohem vhodnější volba nežli přístup strukturovaný. Obrovská výhoda tohoto přístupu plyne již z jeho podstaty, protože při navrhování je možné naprogramovat jakýkoli podprogram v podobě objektu a celkem jednoduše ho zapojit do celkového programu. Další velkou výhodou tohoto přístupu jsou rychle proveditelné změny v programu, protože pokud bude například potřeba změnit vlastnost všech tlačítek v programu, která pocházejí ze stejné třídy, stačí změnit vlastnost pouze v této třídě. Ale přesto tato metoda programování představuje své vlastní nevýhody. Objektově orientované programování může být složitější pro nezkušeného programátora než strukturované programování, pokud jde například o návrh změn nebo o pouhou čitelnost kódu. [8]

Typickými představiteli objektově orientovaného paradigmatu jsou jazyky Java, C++, C#, Python, Perl atd.

1.3 Které paradigma je vhodné pro začátečníky?

Z předchozího srovnání objektově orientovaného a strukturovaného přístupu je zřejmé, že pro začátečníky je vhodnější strukturované paradigma, protože je příhodnější pro malé projekty, s kterými se bude začátečník potýkat rozhodně častěji než s projekty velkými. Zároveň je strukturované programování přehlednější a jednodušší.

Objektově orientované programování vyžaduje mnohem vyšší úroveň abstraktního myšlení než strukturované programování. V praxi se ukazuje, že pokud se začne učit kvalitně strukturovaně, nic se nezmešká a na objektové programování se dá později snadno přeučit. [12]

Nicméně můj názor je takový, že je nejdříve potřeba naučit pomalu úplným základům programování, jako jsou datové typy, větvení, cykly, pole atd., přičemž se začátečníkovi nemusí ještě odkrýt žádný složitější mechanizmus, který dělá ze strukturovaného programování strukturované nebo z objektového objektové. Po vysvětlení těchto základů by začátečník již měl být schopen proniknout do hlubších detailů programování a může si tedy zvolit, ve kterém paradigmatu pokračovat.

2 Programovací jazyky

Nyní se od obecných paradigmat programování přesunu ke konkrétním programovacím jazykům, které můžou být pro začátečníka méně či více vhodné. Některé jazyky dokonce podporují obě již zmíněná paradigmata, takže si začátečník může opravdu vybrat podle jeho zájmu. Programovacích jazyků je nepřeberné množství, a proto zde budou uvedeny pouze ty nejpopulárnější a ty, které jsou obecně považovány za výukově vhodné pro začátečníky.

2.1 Jazyk Pascal

Tento programovací jazyk vytvořil Niklausem Wirthem a jeho primární snahou bylo vytvořit programovací jazyk, ve kterém by bylo jednoduché psát programy pomocí strukturovaného programování a byl by též vhodný pro výuku programování. [5]

Spousta odborníků se shoduje na tom, že je vhodným prvním jazykem pro začínající programátory, protože je do značné míry didaktický. Pokud v něm chce někdo realizovat nějaký kód, zpravidla je jednodušší v Pascalu naprogramovat elegantní řešení než nějaké nešikovné. Nezanedbatelnou výhodou tohoto jazyka je jeho jednoduchost a návaznost na další jazyky, protože obsahuje většinu konstrukcí, se kterými se můžeme setkat i v jazycích jiných. Začátečníci se základní znalostí angličtiny rozhodně v Pascalu ocení jednoduchá klíčová slova, která představují základní anglická slovíčka. [2 s. 17]

Jazyk Pascal je typickým představitelem kompilovaného[1] (překládaného) programovacího jazyka.

2.2 Jazyk C++

Jazyk C++ je objektově orientovanou nadstavbou jazyka C (který je pouze strukturovaný). V jazyce C++ je možné programovat jak objektově, tak též strukturovaně. Je považován za jazyk vyšší úrovně, nicméně povoluje i kroky charakteristické pro jazyky úrovně nižší. Jedná se o jeden z nejpoužívanějších jazyků v současné době. [14]

Výhodou tohoto jazyka je například jeho plná kompatibilita s jazykem C, tudíž je možné jakýkoli zdrojový kód napsaný v jazyce C beze změn umístit do zdrojového kódu jazyka C++. Za další výhodu se dá považovat relativně krátký zdrojový kód oproti ostatním programovacím jazykům. [15] C++ je bohatý komplexní jazyk, avšak není potřeba se jej naučit kompletně celý. Tento jazyk má výhodu hlavně v tom, že svou obecností představuje dobrý základ pro přechod

[1] Kompilovaný jazyk znamená, že před spuštěním programu je potřeba použít kompilátor (překladač), který přeloží zdrojový kód programu do strojového kódu dané platformy. Tato metoda zaručuje velkou rychlost programu, ale je nutné kompilovat program pro každou platformu zvlášť. [9 s. 17]

programátora na další programovací jazyky, nicméně přechod není nutný, protože C++ je v praxi hojně používané. [16 s. 24]

Jazyk C++ je kompilovaný jazyk a jeho předností je rychlost vykonávání operací, ale je tudíž nutné přeložit ho pro každou platformu zvlášť.

2.3 Jazyk C#

Programovací jazyk C# je čistě objektově orientovaný programovací jazyk a je zjednodušenou a vylepšenou verzí jazyka C++. Je také v mnoha ohledech velice podobný jazyku Java. [11]

Pro vytváření a následné spouštění programů v jazyce C# je potřeba prostředí .NET Framework, které se skládá z několika částí. První částí je běhové prostředí Common Language Runtime (CLR), které zajišťuje běh programů přeložených z různých programovacích jazyků do mezijazyka Microsoft Intermediate Language (MSIL). Další částí je knihovna Basic Class Library (BLC), která obsahuje třídy pro ukládání různých druhů dat. Dále toto prostředí obsahuje knihovny pro grafické uživatelské rozhraní, webové služby atd. [11]

Jazyk C# je tedy napůl interpretovaným[2] a napůl kompilovaným jazykem a je tedy velice dobře přenositelný na úkor větší náročnosti.

Programovací jazyk C# se vyznačuje velkým množstvím knihoven pro grafické uživatelské rozhraní a díky tomu také jednoduchým vytvářením grafických prostředí. [3 s. 8]

2.4 Jazyk Java

Jazyk Java je silně objektově orientovaný programovací jazyk, který vychází z jazyka C++, proto s ním má mnoho společného. Oproti jazyku C++ ovšem neobsahuje většinu zbytečně složitých konstrukcí, které při špatném naprogramování způsobovaly problémy. Java je také díky virtuálnímu stroji, který musí být nainstalován na počítači, na kterém je program spouštěn, plně multiplatformní. Java je dnes jedním z nejpopulárnějších programovacích jazyků, z důvodu již zmíněné multiplatformnosti, ale také díky jeho širokému využití. [13]

Java si svou slávu získala jednak v oblasti internetu, protože se v něm dala jednoduše použít, ale také díky její veliké přednosti, kterou je její bezpečnost. Svoje využití má Java v mnoha oblastech, proto je vhodná jak pro běžné aplikace, tak především pro vytváření appletů[3] atd. [10 s. 21]

[2] Interpretované programovací jazyky jsou překládané za běhu programu právě tzv. interpretem, který musí být přítomen na platformě, kde je program spouštěn. Tím, že je program překládán za běhu, se vykonávání programu velice zpomaluje. Nicméně výhodou interpretovaného jazyka je zajištění multiplatformnosti. [9 s. 17]

[3] „Krátké programy, které lze vkládat do webových stránek.“ [10 s. 21]

Java se vyznačuje tím, že není ani interpretovaným ani kompilovaným jazykem, ale něčím mezi, protože zdrojový kód je přeložen pouze do jakéhosi univerzálního mezikódu, který se nazývá byte kód. Tento byte kód je již interpretován na konkrétním stroji pomocí java virtual machine, což je virtuální stroj, který musí být nainstalován v konkrétním zařízení, kde má být program spouštěn. To má za následek rychlejší běh programu než u čistě interpretovaných jazyků a zaručuje to již zmíněnou přenositelnost programu na libovolnou platformu. [10 s. 10]

2.5 Jazyk Python

Python je primárně objektově orientovaný programovací jazyk, který je určen pro efektivní vývoj aplikací. Je to také jeden z jazyků, který je širokou komunitou doporučován jako jazyk pro začátečníky díky jeho přímočaré a jednoduché syntaxi. Nicméně to neznamená, že by v Pythonu nebylo možné vytvořit rozsáhlé aplikace. [17]

Jeho jednoduchost spočívá především v tom, že obsahuje opravdu malé množství klíčových slov, a tudíž je možné začít programovat již po pár minutách učení. Další výhodou pro začátečníky může být fakt, že v Pythonu je možné programovat objektově i strukturovaně a student si tak může vybrat, co mu více vyhovuje. [5]

Jazyk Python je stejně jako jazyk Java napůl interpretovaným a napůl kompilovaným jazykem.

2.6 Který jazyk je vhodný pro začátečníky?

„První programovací jazyk má vliv na celou praxi programátora.“ [2 s. 9] Jazyky Java, C# a C++ jsou dnes sice jedny z nepoužívanějších programovacích jazyků v praxi. Nicméně toto místo si zasloužily nejenom svou jednoduchostí, ale také svým širokým využitím. A právě z tohoto důvodu oba tyto jazyky obsahují konstrukce, které mohou být pro začátečníka lehce matoucí nebo zbytečně složité.

Zajímavým programovacím jazykem je z didaktického hlediska Python, protože ten sice také nebyl primárně vytvořen pro výuku programování, nicméně se podařilo udělat opravdu jednoduchý a přitom široce využitelný programovací jazyk. K jeho výukové vhodnosti pravděpodobně přispěl i fakt, že jeho autor při tvorbě tohoto jazyka částečně vycházel z jazyka ABC, který byl určen právě pro výuku programování. [18]

Jazyk Pascal by se mohl už od prvního pohledu zdát jasným vítězem, protože byl pro účely výuky vytvořen. Vede k dobrým programátorským návykům a dá se z něj lehko navázat na další programovací jazyky. Nicméně si myslím, že mezi Pythonem a Pascalem není moc velký

rozdíl ve vhodnosti pro výuku programování a i přesto, že já sám jsem se nakonec také přiklonil k Pascalu, uznávám i velkou didaktickou hodnotu Pythonu.

Nicméně nesmíme opomenout, že důležité je naučit se programovat a ne naučit se konkrétní programovací jazyk. Pro tyto účely by se tedy měl zvolit jazyk, který vede k dobrým programátorským návykům a naopak nesvádí k obcházení nějakých složitých konstrukcí neelegantním způsobem. První programovací jazyk by měl studenta pouze uvést do problematiky programování a ne ho encyklopedicky učit všechny konstrukce jazyka. [5]

3 Motivace

Velice důležitým prvkem jakékoli výuky je motivace a jinak tomu není ani u programování. Nicméně si myslím, že programování má jednu obrovskou výhodu a tou je jeho schopnost fascinovat žáky naprosto jednoduchými aplikacemi a podnítit v nich iniciativu k samostatnému vymýšlení a tvoření těchto aplikací. U začátečníků se bude jednat o velice nedokonalé (a co se týče využitelnosti v podstatě zbytečné) aplikace, které mnohdy nemají žádný velký význam. Přesto žák bude mít ze svého výtvoru velikou radost a bude mít pocit, že vytvořil něco složitého až do chvíle, kdy se naučí další složitější konstrukce a zjistí, jak bylo jeho původní řešení jednoduché, čímž se mu dostává další samovolné motivace, aby vylepšil svůj stávající program nebo vymyslel úplně nový a dokonalejší.

Někteří autoři knih o programování tvrdí, že dosud velice rozšířená výuka programování na konzolových aplikacích je zastaralá a není motivující. A že již nastala doba, kdy je se začátečníkem možné programovat jednoduché programy využívající grafické rozhraní místo textového. Možné je to právě proto, že jsou dnes již některé jazyky vybaveny jednoduchými funkcemi pro zobrazení grafického rozhraní a je tedy stejně obtížné zobrazit výstup v grafickém rozhraní jako v textovém. [3 s. 8]

Já s tímto přístupem zcela nesouhlasím, a to z několika důvodů. Zaprvé z vlastní zkušenosti bývalého studenta průmyslové školy vím, že pro mě i pro mé spolužáky bylo velice fascinující programovat a samostatně vytvářet jednoduché aplikace právě v textovém prostředí. Samozřejmě tato fascinace postupem času zmizí, ale v tu chvíli již může nastoupit výuka grafického rozhraní. Druhým a z mého pohledu ještě důležitějším důvodem, proč se s tímto postojem neztotožňuji, je fakt, že těmi jazyky, ve kterých je podle Vystavěla vytváření grafického rozhraní stejně obtížné jako textového, jsou myšleny jazyky jako C#, Java atd. A jak již bylo řečeno v předchozích kapitolách, toto nejsou jazyky vhodné pro začátečníky. [3 s. 8]

4 Teorie versus praxe

Většina autorů se shoduje na tom, že programování je praktická činnost, a tudíž by se měla učit převážně praxí [3 str. 9][19 s. 15]. Zejména u začátečníků by měl být poměr praxe k teorii mnohem větší, aby si žáci osvojili základy programování a neměli později problémy vyřešit komplexnější úlohy pomocí několika základních konstrukcí.

Pokud ovšem budeme uvažovat o výchově schopného programátora a ne o nějakých rychlokurzech a podobných pofidérních výukových metodách, je potřeba se zamyslet nad tím, co všechno je ke správnému programování nutné. Budeme tedy spíše směřovat k výuce programování na středních průmyslových školách a případně i na vysokých školách.

Zajímavostí je, že na většině průmyslových škol se začíná právě praxí, a to z toho důvodu, že většina žáku v prvním ročníku ještě nemá zcela jasnou představu o tom, v čem správné programování spočívá a někteří dokonce vůbec netuší, co to programování je. Proto se většinou v prvním nebo druhém ročníku učí nějaký jednoduchý programovací jazyk, který studenty uvede do této problematiky, aby měli na čem stavět. V tomto případě se mají studenti naučit pouze základní prvky a metody programování (proměnná, větvení, cykly, atd.) a rozhodně není účelem tohoto úvodu naučit žáky všem funkcím tohoto jazyka[4].

Po absolvování tohoto úvodního předmětu (případně zároveň s ním) by měla přijít teoretická část. Zde by měla být řádně vysvětlena algoritmizace a datové struktury, zároveň by měla být důkladně probrána číslicová technika (zejména Booleova logika[5]). Dále se již předměty hodně liší podle zaměření programátora nebo školy a týkají se většinou již aplikace programování (počítačové sítě, hardware PC, operační systémy atd.).

Po osvojení těchto základních předmětů pro výuku programování přichází dlouho očekávaná část, kterou je výuka samotného programovacího jazyka. Tento předmět doprovází studenty z oboru programování většinou až do konce studia. Většinou se dokonce nezůstane pouze u jednoho jazyka, aby byli studenti připraveni na různé požadavky budoucích zaměstnavatelů nebo případně vysokých škol.

[4] Právě pro tento úvodní předmět byl vytvořen program Krokovač, který je součástí praktické části této práce a dodržuje již zmíněné předpoklady pro tento předmět. Obsahuje pouze jednoduchou látku pro pochopení základních principů a prvků programování a pro jejich demonstraci byl vybrán jazyk Pascal, protože vede programátory k dobrým návykům.

[5] „Booleova logika je kompletní systém pro logické operace na množině {0,1}.“ [21]

5 Programování i pro malé děti

Kromě plnohodnotných programovacích jazyků, o kterých byly předchozí kapitoly, existují i jazyky, které jsou přímo specializované na výuku programování a jsou dokonce tak jednoduché, že lze většinu z nich vyučovat již na prvním stupni základní školy.

Vývojové prostředí pro takový jazyk je většinou pouze jedno a také se většinou jmenuje stejně jako daný jazyk.

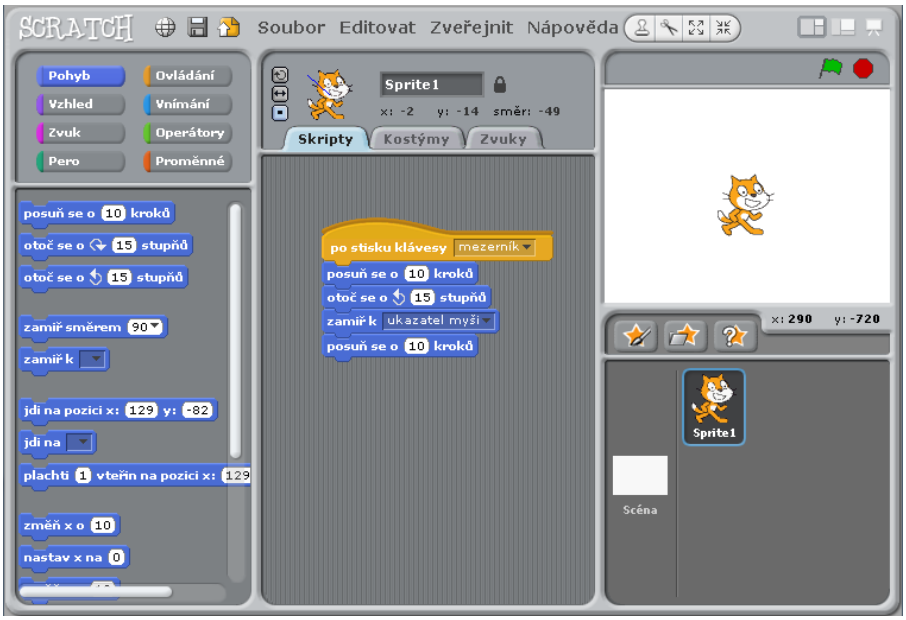
Tyto jazyky se od normálních programovacích jazyků liší především tím, že jejich algoritmus není vytvářen pomocí strukturovaného textu, ale pomocí tzv. vizuálního programování. Tím je myšleno programování pomocí skládání různých diagramů nebo jiných grafických prvků. Většinou není v těchto programovacích jazycích potřeba hledat chyby nebo dávat pozor na překlepy, protože samotný princip grafického programování je nedovolí ani udělat. [6]

Další velkou výhodou těchto jednoduchých jazyků je, že jsou pro malé děti velice zábavné a motivační, protože se jedná převážně o úlohy typu: „Kočička se posune doleva, pak se otočí a posune se doprava.“, přičemž kočička je ve vývojovém prostředí opravdu graficky zobrazena (viz Obrázek 1) a udělá, co jí bylo nařízeno. Takováto úloha by v normálním programovacím jazyku zabrala mnohonásobně více času a vyžadovala by pokročilé znalosti jazyka.

Takto dávají tyto výukové programovací jazyky obrovský základ algoritmickému myšlení velice zábavným a jednoduchým způsobem, aniž by si děti uvědomovaly, že se učí programovat.

5.1 Project Scratch

Vývojové prostředí Scratch je velmi jednoduché a uživatel se v něm může zorientovat během několika minut (viz Obrázek 1). Scratch se především soustředí na vytváření různých her a videí, tedy na grafický výstup programu.



Obrázek 1: Ukázka vývojového prostředí programu Scratch

V pravém horním okně vývojového prostředí je panel, ve kterém se nacházejí grafické prvky, které si můžeme libovolně přidávat nebo odebírat a případně je dokonce algoritmem vykreslovat.

Algoritmus ovládající grafické prvky se sestavuje pomocí bloků, které představují různé příkazy tohoto jazyka. Tyto příkazy jsou rozděleny na několik kategorií, z nichž některé se starají například o pohyb zvoleného grafického prvku nebo o zvukový výstup programu. Tyto bloky do sebe zapadají pomocí svého tvaru jako puzzle, a proto není možné udělat chybu.

Jazyk obsahuje i proměnné, matematické operátory a dokonce i logické operátory, čímž představuje opravdu zajímavý a jednoduchý náhled do světa programování. Nicméně Scratch podporuje i tvorbu vícevláknových programů a posílání zpráv mezi objekty, čímž uvádí již začátečnické uživatele zcela jednoduše do problematiky, ke které se programátoři dostávají až po dlouhé době úporného programování jednovláknových sekvenčních programů. [6]

Velkou motivační výhodou tohoto projektu je také to, že vývojové prostředí obsahuje přímo podporu pro sdílení vytvořených programů, pomocí které může být program nahrán na internet. Programy sdílené na internetu mohou být jednoduše spuštěny přímo ve webovém prohlížeči, protože je možné je spustit pomocí java appletu. [6]

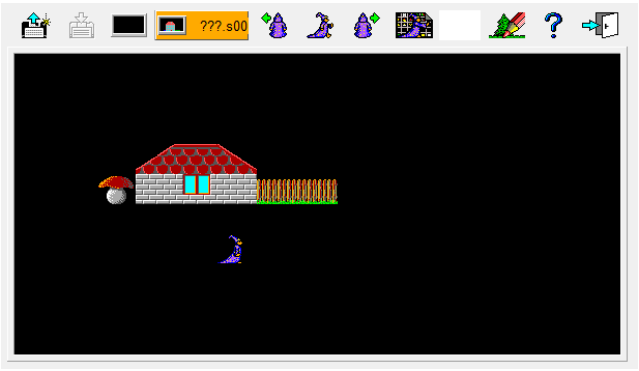
5.2 Projekt Baltík

Program Baltík je multimediální programovací a kreslicí nástroj, který rozvíjí u dětí od 4 let tvořivost a logické myšlení. Syntaxe Baltíka vychází ze syntaxe jazyka C, akorát příkazy nejsou zadávány v textové, ale v grafické formě pomocí bloků. Baltík také nabízí grafický bitmapový editor pro snadné vytváření grafických animací. [20]

Vývojové prostředí Baltíka nabízí čtyři režimy, které se dělí podle věkové náročnosti. Prvním je režim „Skládat scénu“, který je vhodný pro děti od 4 let. V tomto režimu mají děti k dispozici paletu obsahující velké množství dílečků, ze kterých se dá sestavit docela různorodá scéna (viz Obrázek 2).

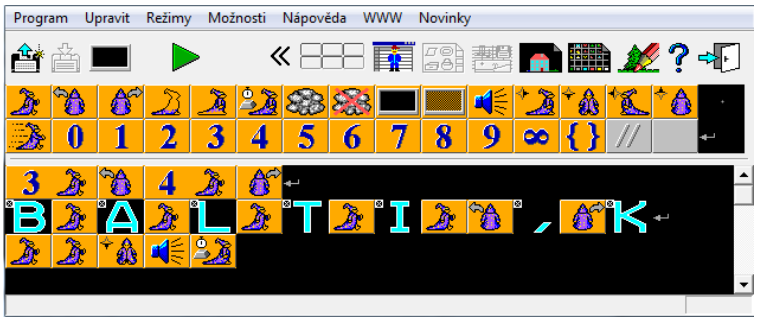
Obrázek 2: Ukázka plochy pro skládání scény

Druhý režim „Čarovat scénu“ je vhodný pro děti od 6 let a již je připravuje na pozdější princip programování scény. V tomto režimu totiž mají k dispozici interaktivní postavičku kouzelníka Baltíka (viz Obrázek 3), se kterou smějí pohybovat pomocí tlačítek a jednotlivé objekty mohou umisťovat pouze pomocí této postavičky na místo, kde právě stojí.



Obrázek 3: Ukázka kouzelníka Baltíka uprostřed plochy

Třetí režim „Programování Baltíka“ je určen pro děti ve věku od 6 do 13 let. Tento režim již ukazuje dětem jednoduchou formou základní principy programování. V tomto režimu mají děti k dispozici základní příkazy pro pohyb postavičky Baltíka (viz Obrázek 4), což jim umožňuje vytvářet jednoduché animace, pohádky a prezentace.



Obrázek 4: Ukázka palety s příkazy (horní část) a jednoduchého programu (dolní část)

Posledním režimem je „Klasické programování“, který je určen pro děti od 9 let. Tento režim nabízí širokou škálu příkazů (viz Obrázek 5), které korespondují s jazykem C a dají se díky tomu sestavit již relativně složité aplikace.

Obrázek 5: Ukázka palety s příkazy v horní části a programu v dolní části

5.3 Scratch versus Baltík

Velkou výhodou projektu Scratch je to, že je dostupný zdarma, zatímco projekt Baltík je zdarma pouze v demoverzi, ve které se nedají vytvořené programy ukládat. Další výhodou programu Scratch je jeho značná jednoduchost a velice intuitivní ovládání i pro malé děti. Baltík je na ovládání o něco méně intuitivní, nicméně jeho rozdělení na režimy podle věku nabízí větší věkovou flexibilitu.

6 Kdy začít vyučovat programování?

Programování se dnes dá využít ve všech odvětvích a nabízí se tedy otázka, zda by programování neměly alespoň z části okusit již děti na základní škole. Samozřejmě nemůžeme očekávat, že se děti ze základní školy naučí opravdu programovat, nicméně pokud by se jim algoritmické a logické myšlení rozvíjelo již na základní škole pomocí dětských programovacích jazyků, byly by mnohem lépe připraveny na programování na středních školách. Algoritmické a logické myšlení ale není užitečné pouze pro programátory, naopak je využitelné i v běžném životě a v jiných profesích.

Bohužel rámcový vzdělávací program pro základní školy programování nezahrnuje povinně do běžné výuky informačních a komunikačních technologií. Algoritmizace je sice několikrát zmíněna v souvislosti s informačními a komunikačními technologiemi, ale nenachází se mezi očekávanými výstupy, které jsou jediné závazné pro každou školu. Je samozřejmě možné zařadit programování do výuky jako takové i přesto, že ve výstupech není. Častěji se ale stává, že je programování vyučováno v rámci povinně volitelných nebo případně volitelných předmětů. To je ale v režii samotné školy, a tudíž nemůžeme po příchodu žáků na střední školu počítat s tím, že již všichni znají elementární prvky a postupy programování. [23 s. 34, 35, 36]

Základy programování se tedy ve většině případů probírají až na některých středních školách, většinou s technicky zaměřenými obory. Tohoto faktu jsem se držel i při vytváření programu v praktické části bakalářské práce, který je určen pro úplné začátečníky a může sloužit jako pomůcka, která žákům ulehčí pochopit základní konstrukce programování nebo jako materiál pro samostudium.

7 Popis programu

Program byl nazván Krokovač, neboť jeho hlavní funkcí je zpracování programů napsaných v jazyce Pascal **krok za krokem**, přičemž právě uživatel rozhoduje o tom, kdy se přejde na další krok. V každém kroku je vidět, jak se změnily proměnné a případně, zda bylo něco vypsáno na standardní výstup. Velkou výhodou tohoto programu je také to, že krokování je zobrazováno zároveň jak na zdrojovém kódu, tak na vývojovém diagramu.

Před příklady, které představují novou látku, se vždy nachází stránka s vysvětlenou teorií potřebnou pro pochopení příkladu. Teorie v programu není rozebírána podrobně, ale jsou vysvětleny všechny důležité prvky jazyka.

Program je zaměřen na úplné začátečníky programování, tudíž je největší část programu věnována větvení programu a cyklům, což jsou základní znalosti potřebné pro programování. Dále jsou mezi vysvětlovanou látku zahrnuta pouze ta témata, která se používají v převážné většině jazyků a nejsou pouze výhradou jazyka Pascal. Nepovažuji dnes totiž jazyk Pascal za použitelný v praxi, ale uznávám jeho obrovskou hodnotu pro začátečníky díky jeho jednoduchosti a názornosti.

7.1 Obecná forma programu a jeho ovládání

Program se svou formou podobá knize. První podobností je interaktivní obsah (viz Obrázek 6), který se zobrazí při spuštění programu a slouží jako rozcestník. Obsah je rozdělen na šest základních kategorií, které jsou pouze orientační a při kliknutí na jednu z nich se zobrazí její položky. Na tyto jednotlivé položky je také možno kliknout, čímž se dostaneme na příslušnou stránku programu.

Obrázek 6: Ukázka obsahu s rozbalovacími kategoriemi v programu

Další podobností jsou stránky programu, které jsou seřazeny za sebou. Na další/předchozí stránku se uživatel může přesunout pomocí šipek doleva a doprava na klávesnici, nebo kliknutím na šipky doleva a doprava na dolním ovládacím panelu programu (viz Obrázek 7).

Obrázek 7: Ukázka dolního ovládacího panelu

Z každé stránky je možné se kliknutím na tlačítko „Zpět na Obsah“ (viz Obrázek 7) dostat zpět na stránku s obsahem.

V programu jsou dva základní druhy stránek. Jednak jsou to teoretické stránky, které obsahují pouze text, jenž vysvětluje novou teorii. A jednak praktické stránky, které obsahují příklady, jež pomáhají pochopení a procvičují právě probranou teorii.

Dolní ovládací panel zůstává pro oba druhy stránek stejný, až na jedno tlačítko, které mění svou funkci podle toho, na které stránce se zrovna nacházíme. Pokud se uživatel nachází na stránce s teorií, v dolním panelu je tlačítko „Otevřít v PDF“ (viz Obrázek 8), pomocí něhož si může uživatel zobrazenou teorii otevřít ve vlastním PDF prohlížeči, tedy v novém okně. Výhoda je především v tom, že pokud si uživatel takto teorii otevře v novém okně, může ji mít jako pomůcku vedle programu při procházení příkladu, který se k ní vztahuje.

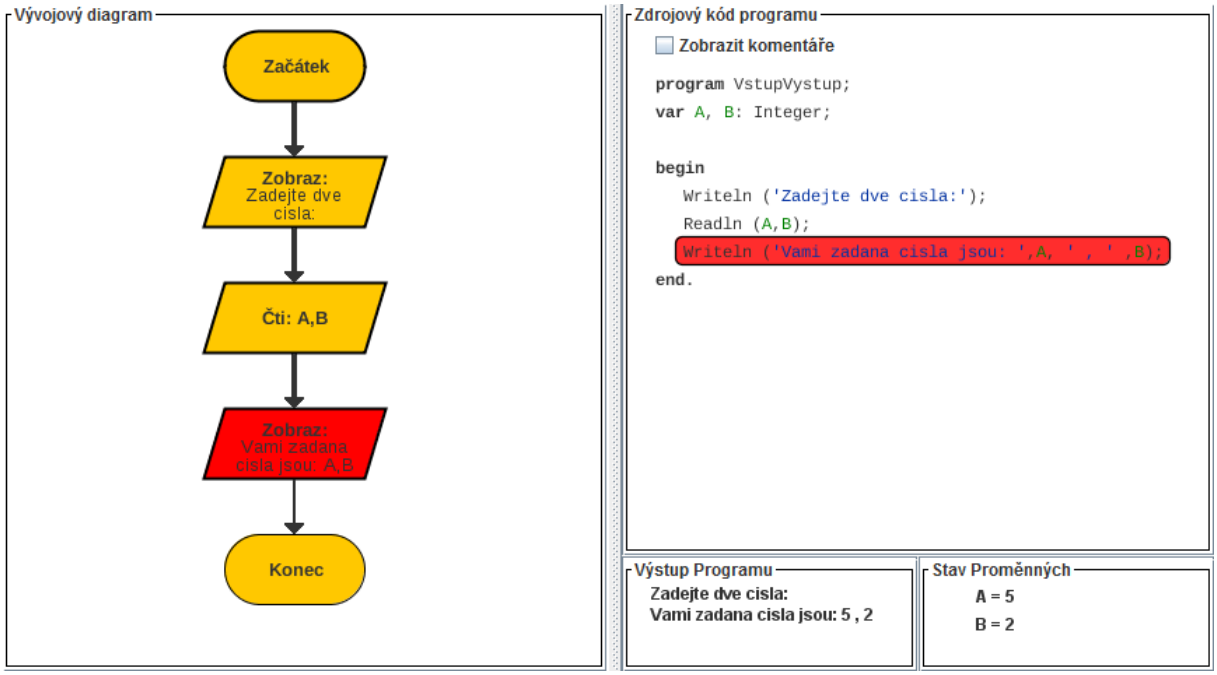
Obrázek 8: Ukázka dolního ovládacího panelu na stránce s teorií

Pokud se uživatel nachází na stránce s příkladem, je v dolním panelu tlačítko „Otevřít zadání“ (viz Obrázek 9), pomocí něhož může otevřít textový soubor ve vlastním PDF prohlížeči. V tomto souboru se nachází samotné zadání příkladu a následně i rozbor řešení.

Obrázek 9: Ukázka dolního ovládacího panelu na stránce s příkladem

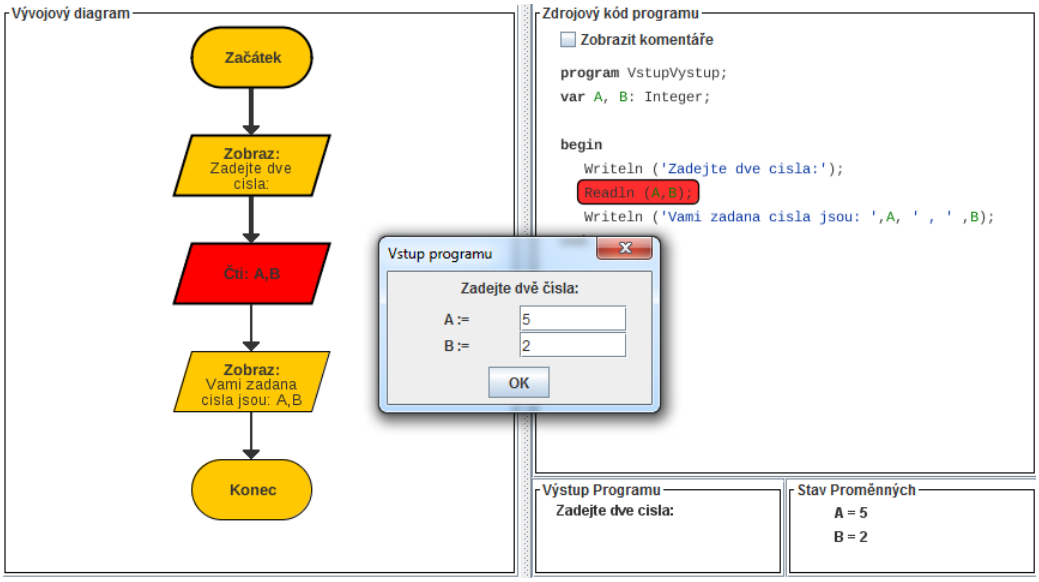
7.2 Příklady v programu

Jak již bylo zmíněno, příklady si může uživatel libovolně krokovat, přičemž může sledovat aktuální výstup a hodnoty proměnných. Současně je možné krokování sledovat jak na vývojovém diagramu, tak na zdrojovém kódu (viz Obrázek 10).



Obrázek 10: Ukázka příkladu v programu

Také vstupní hodnoty mohou být uživatelem zadány dle vlastního uvážení pomocí dialogového okna (viz Obrázek 11), které se zobrazí ve chvíli, kdy se program dostane do kroku, ve kterém má číst ze standardního vstupu.



Obrázek 11: Ukázka dialogového okna pro zadání vstupu

Ve zdrojovém kódu si uživatel může zapnout/vypnout zobrazení komentářů (viz Obrázek 12), ve kterých jsou vysvětleny nové příkazy anebo případně složitější algoritmy.

Zdrojový kód programu

Zobrazit komentáře

```
program VstupVystup;   //hlavička programu
var A, B: Integer;   {deklarace proměnných A,B celočíselného typu}

begin   //začátek bloku
   Writeln ('Zadejte dve cisla:');   {vypsání textu do konzole a odřádkování}
   Readln (A,B);   {přečtení proměnných z konzole a odřádkování}
   Writeln ('Vami zadana cisla jsou: ',A, ' , ' ,B);   {vypsání textu a proměnných do konzole a odřádkování}
end.   //konec bloku
```

Obrázek 12: Ukázka zapnutých komentářů ve zdrojovém kódu

Samotné krokování je možné ovládat několika způsoby. Zaprvé lze libovolně kliknout na jakýkoli objekt ve vývojovém diagramu, čímž může uživatel některé kroky přeskočit. V případě, že uživatel přeskočil zadávání vstupních hodnot, jsou použity hodnoty defaultní. Stejným způsobem je možné libovolně kliknout na jakýkoli řádek ve zdrojovém kódu. Zadruhé lze ovládat krokování šipkami nahoru a dolu na klávesnici, čímž se uživatel posune na následující nebo předchozí krok. A zatřetí je program možno ovládat myší, kliknutím na šipku nahoru a dolu na ovládacím panelu (viz Obrázek 13), který je umístěn v dolní časti programu.

Obrázek 13: Ukázka šipek nahoru a dolu v dolním ovládacím panelu

U příkladů, kde se vykytuje nějaký druh cyklu, je ovládání omezeno. Nelze se vrátit o krok zpět a nelze klikat myší na libovolné objekty v diagramu nebo zdrojovém kódu. Jinými slovy, lze se dostat pouze o krok vpřed. Pokud by byly tyto funkce ovládání povoleny, nebylo by jasné, kolikrát se má cyklus vykonat, protože v některých příkladech se o počtu projití cyklem rozhoduje až uvnitř cyklu.

7.3 Obsah látky v programu

Každá látka v programu je vysvětlena až ve chvíli, když je potřeba pro další příklad. Příklady jsou seřazeny tak, aby na sebe látka co nejvíce navazovala a aby obtížnost pomalu stoupala.

Na začátku programu je vysvětleno, co je to algoritmus, jeho vlastnosti a možnosti vyjádření, kde jsou podrobněji popsány dva způsoby použité v tomto programu (vývojovým diagram a zdrojový kód).

V další kapitole jsou vysvětleny základní prvky programu, jako je příkaz, komentář, proměnná atd. Dále jsou vyloženy příkazy Readln a Writeln (Write). Vysvětlení příkazu Read bylo

z důvodu větší náročnosti na pochopení ponecháno až na chvíli, kdy jej bude potřeba použít. Následuje vysvětlení přiřazovacího příkazu a číselných datových typů, které jsou zde sice popsány všechny, nicméně v programu jsou používány pouze Integer a Real. Poté je zde vysvětlen semilogaritmický zápis čísel v Pascalu, nicméně v Krokovači jsou hodnoty reálného datového typu vypisovány klasickým způsobem s desetinou tečkou pro dosažení lepší přehlednosti a názornosti. Tato kapitola byla pouze přípravou na kapitolu další, jíž jsou Číselné operace.

Následuje vysvětlení neúplného podmíněného příkazu, složeného příkazu a úplného podmíněného příkazu. Dále jsou popsány aritmetické funkce, z nichž ale větší část popsaných není v Krokovači přímo použita. Poté je vyložen další druh větvení a tím je příkaz case a také logické operace.

Další probíranou látkou jsou cykly. Začíná se vysvětlením While – cyklu, dále je vysvětlen Repeat – cyklus a For – cyklus.

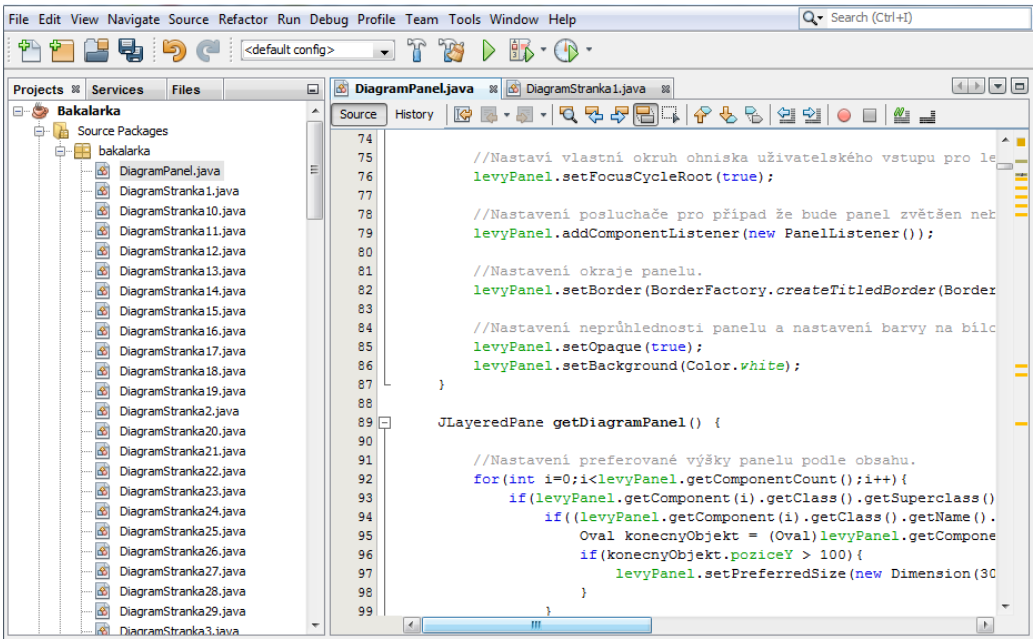
Poslední částí programu jsou nečíselné datové typy, které začínají datovým typem Char, po kterém je vysvětlen příkaz Read, protože jej již bude potřeba použít. Následuje datový typ Boolean, typ pole, vysvětlení konstant a datový typ String.

8 Vývojové prostředí

Program je naprogramován v jazyku Java, který jsem si jako programovací jazyk vybral hned z několika důvodů. Jeden z hlavních byl ten, že v Javě je velké množství knihoven a funkcí pro práci s grafickými prvky, což je skoro jediné, co bylo v mém programu potřeba. Dalším důvodem je, že objektově orientované programování mi je bližší a zdá se mi pro tento typ programu vhodné.

Vzhledem k tomu, že tato úloha nevyžaduje žádné zvláštní funkce vývojového prostředí, jsem si vybral populární vývojové prostředí NetBeans (viz Obrázek 14), ve kterém jsem již dříve pracoval. Toto prostředí je open-source a je jednoduché na ovládání. Jeho velkou výhodou je také spousta funkcí pro usnadnění práce.

Protože se jedná o velice rozsáhlé vývojové prostředí s mnoha funkcemi, nebudou zde popsány zdaleka všechny, ale pouze ty nejdůležitější a hlavně ty, které byly využity při tomto projektu.

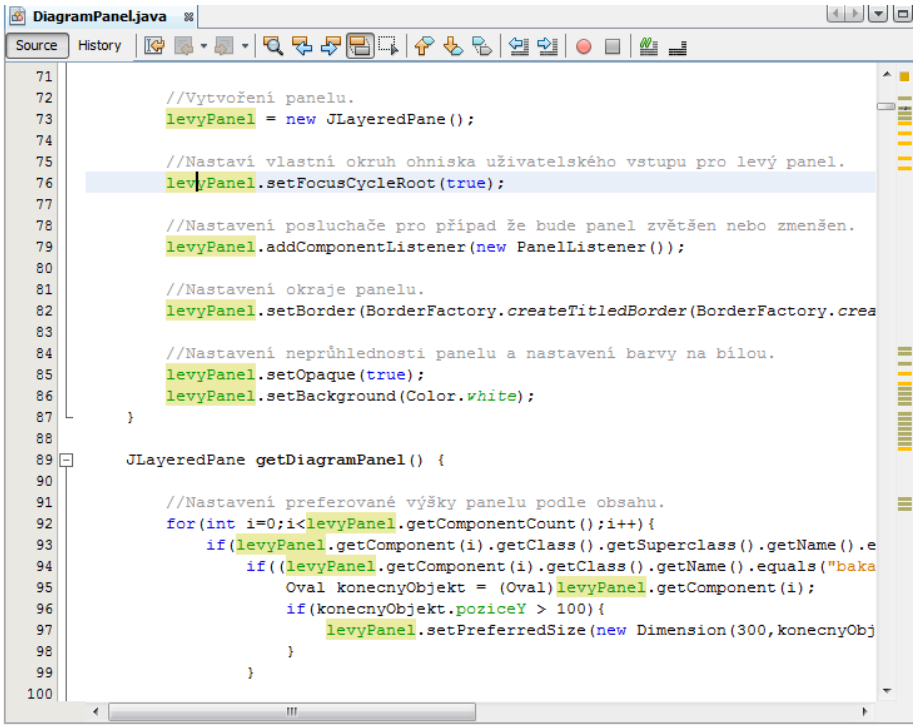


Obrázek 14: Vývojové prostředí NetBeans

V levém panelu programu se nachází všechny soubory a tedy hlavně všechny třídy obsažené v daném projektu (viz Obrázek 14). V pravém panelu (viz Obrázek 14) je možné zobrazit a měnit obsah vybraných tříd.

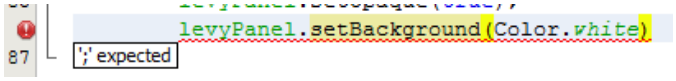
Pravý panel má mnoho prvků usnadňujících práci programátora. Ve zdrojovém kódu jsou barevně rozlišena klíčová slova, znakové řetězce, názvy proměnných atd., což je dnes samozřejmostí průměrného vývojového prostředí.

Velice přehledným činí zdrojový kód funkce, která při umístění kurzoru na nějakou třídu, proměnnou nebo metodu zvýrazní, kde všude se tento prvek nachází (viz Obrázek 15).



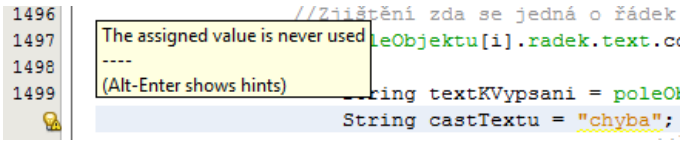
Obrázek 15: Ukázka zvýraznění označeného prvku v prostředí NetBeans

Ve zdrojovém kódu jsou zvýrazněny syntaktické chyby, u kterých je možno zobrazit popis, o jakou chybu se jedná (viz Obrázek 16) a v některých případech je dokonce nabídnut i návrh opravy.



Obrázek 16: Upozornění na chybu v prostředí NetBeans

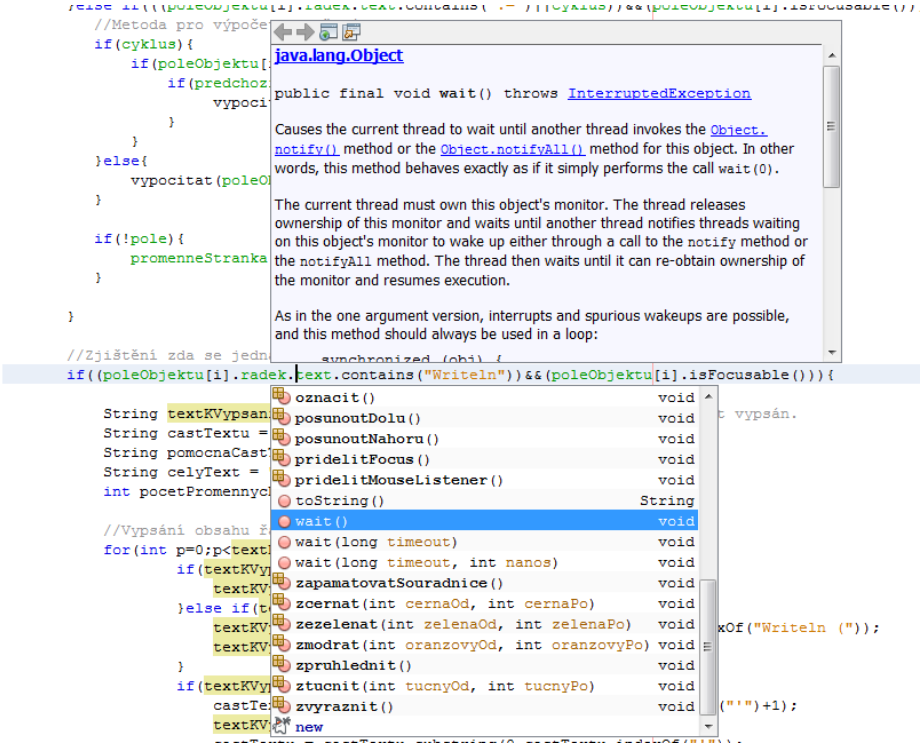
Dále umí vývojové prostředí upozorňovat na místa, kde sice nedošlo k syntaktické chybě, ale něco není úplně v pořádku. Jedná se například o upozornění, že hodnota proměnné nebude nikdy použita a je tedy zbytečná (viz Obrázek 17).



Obrázek 17: Upozornění v prostředí NetBeans

Na pravé straně panelu se nachází lišta, na které jsou barevné značky (viz Obrázek 15). Tato lišta představuje celou třídu a jednotlivé značky znamenají řádky, na kterých je potřeba na něco upozornit. Červené značky znamenají chybu, žluté upozornění a hnědé další výskyt právě označeného prvku.

Velice užitečnou funkcí pro začátečníka i pro pokročilého programátora, je funkce pro kompletaci kódu. V místě, kde se nachází kurzor, nabídne tato funkce všechny možnosti, jak by mohl kód pokračovat a pro každou položku ze seznamu nabídnutých pokračování zobrazí navíc Java dokumentaci, je-li pro ten prvek dostupná (viz Obrázek 18).

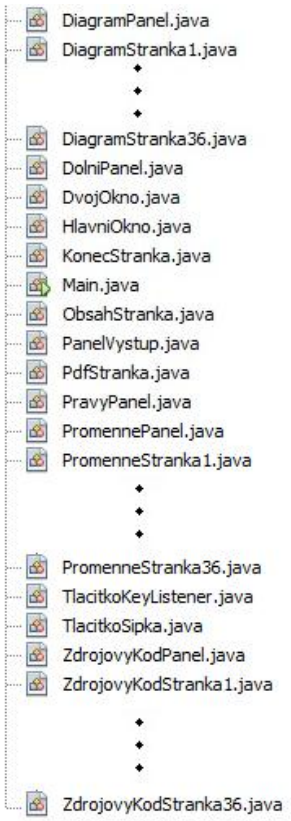


Obrázek 18: Ukázka automatického doplňování kódu s java dokumentací

Další, již pokročilejší funkcí je debugger, pro který je možné ve zdrojovém kódu nastavit breakpointy, u kterých se debugger zastaví a mezi kterými je možné program krokovat. Je možné si nastavit zobrazení určitých proměnných, a tak kontrolovat jejich aktuální stav.

9 Zdrojový kód programu

Program se skládá z velkého množství tříd (viz Obrázek 19). Ty, které jsou vidět na obrázku v hlavní větvi, jsou třídy základní a některé obsahují vlastní podtřídy kvůli jejich specifickému účelu.



Obrázek 19: Ukázka hlavních tříd programu

Naprostá většina tříd souvisí s příklady v programu, neboť každý příklad musí mít tři konkrétní třídy („DiagramStranka“, „PromenneStranka“ a „ZdrojovyKodStranka“). Tyto třídy jsou potomky abstraktních tříd „DiagramPanel“, „PromennePanel“ a „ZdrojovyKodPanel“, které v sobě mají algoritmus společný pro všechny příklady.

Naopak pro stránky s teorií je určena pouze jedna třída „PdfStranka“, která je pro všechny stránky společná, neboť jí je pouze pomocí parametru předána adresa obrázku, představujícího PDF stránku.

Všechny tyto stránky obsažené v programu jsou vytvořeny a seřazeny ve třídě „HlavniOkno“, která je zároveň zodpovědná za zobrazení a nastavení hlavního okna.

Důležitou třídou je také „DolniPanel“, která má za úkol zobrazit a nastavit dolní ovládací panel, včetně tlačítek a jejich funkcí.

Aby bylo možné program ovládat také klávesnicí, nachází se zde třída „TlacitkoKeyListener“, která představuje posluchač stisknutí kláves. Tento posluchač je přidělován různým objektům v programu.

9.1 Stránkování programu

Všechny stránky i se svým obsahem jsou vytvořeny již při spouštění programu, tudíž při samotném procházení programem se vytvořené objekty už pouze zobrazují. Stránky jsou po svém vytvoření vloženy do pole v pořadí, v jakém jsou zobrazovány v programu. Všechny následující kódy, které budou popsány v této kapitole, se nacházejí ve třídě s názvem „HlavniOkno“, která představuje hlavní okno celé aplikace.

Při každé změně stránky se nejdříve z hlavního okna odebere aktuální stránka a poté se do něj vloží stránka, která má být právě zobrazena. Při zobrazování nové stránky je ale nutné zjistit, o jaký druh stránky se jedná (teorie, příklad, obsah, poslední stránka). Druh stránky lze jednoduše zjistit z názvu třídy a poté již stačí provést příslušné metody. Většina těchto metod se týká tlačítek na dolním panelu, neboť ty nemá samotná stránka ve vlastní režii, a proto je zde nutné je explicitně upravit (viz Obrázek 20).

```
case "bakalarka.DvojOkno":
    dvojOkno = (DvojOkno)poleStranek[poradi];
    //Nastavení rozdělovače doprostřed.
    dvojOkno.vystreditRozdelovac();
    //Odebrání tlačitka pro otevírání pdf souborů.
    dolniPanel.odebraniTlacitkaPDF();
    //Vystředění tlačítka pro otevření pdf souborů se zadáním.
    dolniPanel.odebraniTlacitkaZadani();
    dolniPanel.premistitTlacitkoZadani(dvojOkno.levyPanel.adresaZadani);
    //Přenastavení tooltip textu pro tlačítka.
    dolniPanel.zmenitNapisTlacitekNaDvojOkno();
    //Vystředění tlačítka pro návrat na obsah.
    dolniPanel.odebraniTlacitkaObsah();
    dolniPanel.premistitTlacitkoObsah();
    break;
```

Obrázek 20 Ukázka metod, které se provedou v případě zobrazení stránky s příkladem

Další důležitou operací při přepínání stránky je předání ohniska uživatelského vstupu[6] (dále již jen zaměření) právě zobrazované stránce. Tato stránka totiž ve chvíli, kdy dostane zaměření, jej předá dalšímu objektu uvnitř sebe samotné. K čemu to je dobré, bude popsáno v dalších kapitolách.

Důležité je také udržování některých grafických prvků na svém místě i přesto, že se okno zmenšuje či zvětšuje. Pro tento účel je oknu přidělen posluchač s názvem „PanelListener“. Tento

[6] Ohnisko uživatelského vstupu (zaměření) neboli anglicky focus je mechanismus, který určuje, který z objektů v programu bude přijímat události vstupu z klávesnice. Jinými slovy, pokud chceme, aby nějaký objekt reagoval na zmáčknutí klávesy na klávesnici, musí tento objekt být právě vlastníkem zaměření. Zaměření může vlastnit pouze jeden objekt. [22]

posluchač má jedinou metodu „componentResized“, která se vykoná v případě, že bylo okno zvětšeno nebo zmenšeno. V metodě je potřeba rozlišit, o jaký druh stránky se jedná a podle toho umístit příslušné komponenty tam, kam patří (viz Obrázek 21).

```
switch (poleStranek[stranka].getClass().getName()) {
    case "bakalarka.DvojOkno":
        dvojOkno = (DvojOkno)poleStranek[stranka];

        //Nastavení rozdělovače dvojOkna doprostřed.
        dvojOkno.vystreditRozdelovac();

        //Vystředění tlačítka pro otevření pdf souborů se zadáním.
        dolniPanel.odebraniTlacitkaZadani();
        dolniPanel.premistitTlacitkoZadani(dvojOkno.levyPanel.adresaZadani);

        //Vystředění tlačítka pro návrat na obsah.
        dolniPanel.odebraniTlacitkaObsah();
        dolniPanel.premistitTlacitkoObsah();

        //Zobrazení posuvníků v panelech v případě potřeby.
        dvojOkno.pravyPanel.promenneStranka.zmenitVelikost();
        dvojOkno.pravyPanel.panelVystup.zmenitVelikost();

        break;
```

Obrázek 21: Ukázka metod, které se provedou v případě změny rozměrů okna

Většinou se jedná o vystřeďování tlačítek v dolním panelu anebo vystřeďování dalších komponent příslušné stránky. Dále se zde můžou objevit i metody pro změnu preferované velikosti nějakého panelu podle jeho obsahu. Preferovaná velikost těchto panelů je měněna, protože se podle ní řídí zobrazení posuvníků. Respektive, jestliže preferovaná velikost není plně zobrazena, zobrazí se posuvníky, které uživateli umožní prohlížet celý obsah panelu i přes to, že je zobrazena jenom jeho část. Nicméně v této chvíli se obsah panelu nezmění, ale přesto je potřeba metodu provést, protože je naopak možné, že se zmenšením okna změnila samotná velikost panelu, a tudíž je potřeba přehodnotit, zda se mají zobrazit posuvníky.

9.2 Dolní ovládací panel

Dolní ovládací panel (viz Obrázek 22) zůstává po celou dobu běhu programu zobrazen, mění se pouze jeho komponenty v závislosti na druhu zobrazené stránky. Všechny následující kódy, které budou popsány v této kapitole, se nacházejí ve třídě s názvem „DolniPanel“, která představuje ovládací panel v dolní části programu.

Obrázek 22: Ukázka dolního ovládacího panelu

Tato třída sice vytváří dolní panel, ale naprostá většina kódů této třídy se stará spíše o vytvoření a nastavení tlačítek v tomto panelu. U tlačítek nastavuje jejich umístění, velikost, text, ikonu a podobně.

Důležité je zde nastavení toho, aby tlačítko nemohlo získat zaměření. Tento postup byl zvolen z toho důvodu, že zaměření je použito na stránkách s příklady, kde vždy některý z objektů příkladu musí vlastnit zaměření. Nicméně defaultní nastavení Javy je takové, že pokud uživatel klikne myší na tlačítko, získá toto tlačítko zaměření, což by narušovalo algoritmus použitý v příkladech.

Dále tato třída poskytuje velké množství metod, které slouží k odebrání, přidání a vystředění jednotlivých tlačítek. Tyto metody se využívají zejména při již popsaném stránkování programu.

Posluchač „TlacitkoActionListener“

Nejdůležitější částí třídy „DolniPanel“ je nastavení posluchače pro stisknutí tlačítek, který má název „TlacitkoActionListener“ a má jedinou metodu s názvem „actionPerformed“, která je vykonána v případě, že bylo tlačítko s tímto posluchačem stisknuto. Tento posluchač je společný pro všechna tlačítka v dolním panelu, ale pomocí konstruktoru mu je zadán parametr typu String, který slouží k identifikaci tlačítka.

Uvnitř posluchače je potřeba rozlišit nejen které tlačítko vyvolalo událost, ale také jaký druh stránky je zrovna zobrazen, aby bylo možné nastavit funkci tlačítek nahoru a dolu. Na začátku metody „actionPerformed“ je tedy nutné nejprve zjistit, kdo je vlastníkem zaměření, protože jak už bylo zmíněno výše, v programu má zaměření zásadní roli a dá se podle jeho vlastníka jednoznačně určit, o jakou stránku se jedná.

Dále je potřeba rozlišit druh tlačítka, na což je použit parametr „napisTlacitka“, který slouží jako identifikátor. V případě tlačítka s šipkou dolu je potřeba rozlišit, zda se jedná o stránku s teorií nebo příkladem. V případě, že se jedná o stránku s teorií, musí se posunout pouze vertikální posuvníky směrem dolu. Tlačítko nahoru se chová velice obdobně.

Druhý případ, kdy se jedná o stránky s příklady, bude popsán až v dalších kapitolách současně s algoritmem, který se těchto stránek bude týkat.

Tlačítka doleva a doprava neboli „předchozí stránka“ a „následující stránka“ využívají v posluchači metod z již zmíněné třídy „HlavníOkno“. Tyto metody zobrazí předchozí nebo následující stránku.

Dalším tlačítkem, pro který je určen tento posluchač, je tlačítko pro otevírání stránky s teorií v PDF prohlížeči. Vzhledem k tomu, že je požadováno po operačním systému, aby otevřel PDF soubory, musí se tyto soubory nacházet mimo hlavní soubor programu s koncovkou

.jar. Tento hlavní soubor je totiž variantou zabaleného souboru, z něhož operační systém nedokáže soubory spouštět. Proto musely být PDF soubory umístěny do složky s programem, ale ne do samotného programu. Nyní nastává druhý problém – jak zajistit, aby se adresa k PDF souborům dynamicky měnila podle toho, kde je celý program uložen. Tento problém je vyřešen tak, že je možné zjistit adresu, kde se nachází spouštěcí soubor programu, a dále je již pouze potřeba upravit textový řetězec představující adresu ke spouštěcímu souboru a to takovým způsobem, aby představoval adresu k příslušnému PDF souboru (viz Obrázek 23).

```
//Nalezení cesty k adresáři odkud je jar. soubor spouštěn.
slozka = new File(Main.class.getProtectionDomain()
        .getCodeSource().getLocation().toURI()
        .getPath()).getParentFile();

//Upravení cesty složky tak aby obsahovala cestu k pdf souboru.
soubor = new File(slozka +  File.separator + "pdf"
        + File.separator + adresaPDF);
```

Obrázek 23: Nalezení cesty k PDF souborům

Ve chvíli, kdy je zjištěna adresa k samotnému PDF souboru, je třeba ještě zajistit jeho spuštění. Při spouštění souboru může nastat několik problémů, které ho neumožní. Z tohoto důvodu je potřeba nejprve zjistit, zda soubor opravdu existuje. Pokud existuje, je dále potřeba zjistit, zda má aplikace práva spustit PDF soubor. Jestliže aplikace tato práva má, pak již nic nebrání ve spuštění souboru. V případě těchto problémů je vhodné uživatele upozornit zobrazením dialogového okna.

Tlačítko otevírající zadání v příkladech se chová totožně jako tlačítko pro otevírání teorie v PDF prohlížeči.

Posledním tlačítkem je tlačítko pro návrat na obsah. Zde je znovu užito metody z třídy „HlavníOkno“ pro zobrazení nové stránky.

9.3 Stránka s obsahem

Všechny následující kódy, které budou popsány v této kapitole, se nacházejí ve třídě s názvem „ObsahStranka“, která představuje stránku s obsahem.

Samotná stránka je od chvíle svého zobrazení vlastníkem zaměření a z toho důvodu musí mít přidělen posluchač stisknutí klávesy na klávesnici s názvem „TlacitkoKeyListener“, který se postará o ovládání programu pomocí šipek na klávesnici.

Dále tato třída obsahuje podtřídu s názvem „Text“, která představuje jeden samostatný nápis v obsahu.

V konstruktoru třídy „Text“ je použit parametr udávající typ nápisu, pro rozdělení nápisů dle jejich typu (kategorie, odkaz, nadpis, atd.), aby bylo možné nastavení určitých atributů textu

(velikost písma, barva písma, atd.). Některým nápisům, které mají být interaktivní, se vytvoří pozadí ve tvaru oválného obdélníku, které se zobrazuje při najetí kurzoru myši na nápis.

Interaktivním nápisům je přidělen posluchač myši s názvem „NovyMouseListener“. Dvě metody tohoto posluchače se starají o to, aby se při najetí kurzoru myši na text zobrazilo pozadí. A třetí metoda se vykoná v případě, že bylo na nápis kliknuto levým tlačítkem myši. Tato metoda je oproti předchozím dvěma velice rozsáhlá. V první řadě se v ní rozlišuje, zda se jedná o kategorii nebo o odkaz na stránku. V případě, že se jedná o odkaz na stránku, je pouze pomocí dvou metod z třídy „HlavníOkno“ odebrána stránka s obsahem a zobrazena stránka na kterou odkazoval nápis.

Obrázek 24: Ukázka rozbalené kategorie „Jednoduché příkazy“

Mnohem složitější je případ, kdy bylo kliknuto na kategorii (viz Obrázek 24). V tomto případě je potřeba pomocí cyklu projet celé pole s odkazy v této kategorii. V cyklu se zjistí, zda jsou již odkazy zobrazeny. Pokud jsou zobrazeny, znamená to, že se musí kategorie sbalit neboli že se mají odkazy zneviditelnit a další kategorie posunout nahoru (viz Obrázek 25). V případě, že odkazy zobrazeny nejsou, znamená to, že se má kategorie rozbalit neboli zobrazit odkazy a následující kategorie posunout dolů.

```
}else if(typTextu.equals("kategorie")){
    //Zobrazení odkazů pro tuto kategorii a posunutí zbytku.
    for(int i=0;i<poleOdkazu.length;i++){
        if(poleOdkazu[i].napis.isVisible()){
            poleOdkazu[i].napis.setVisible(false);
            poleOdkazu[i].zpruhlednit();
            for(int j=poradiTetoKategorie+1;j<poleKategorii.length;j++){
                poleKategorii[j].pozadi.setBounds(poleKategorii[j]
                        .pozadi.getBounds().x,poleKategorii[j].pozadi.getBounds().y
                        - 27,poleKategorii[j].pozadi.getBounds().width,
                        poleKategorii[j].pozadi.getBounds().height);
                poleKategorii[j].napis.setBounds(poleKategorii[j].napis.getBounds().x,
                        poleKategorii[j].napis.getBounds().y - 27,
                        poleKategorii[j].napis.getBounds().width,poleKategorii[j]
                        .napis.getBounds().height);
                for(int k=0;k<poleKategorii[j].poleOdkazu.length;k++){
                    poleKategorii[j].poleOdkazu[k].pozadi.setBounds(poleKategorii[j]
                            .poleOdkazu[k].pozadi.getBounds().x,poleKategorii[j]
                            .poleOdkazu[k].pozadi.getBounds().y - 27,poleKategorii[j]
                            .poleOdkazu[k].pozadi.getBounds().width,poleKategorii[j]
                            .poleOdkazu[k].pozadi.getBounds().height);
                    poleKategorii[j].poleOdkazu[k].napis.setBounds(poleKategorii[j]
                            .poleOdkazu[k].napis.getBounds().x,poleKategorii[j]
                            .poleOdkazu[k].napis.getBounds().y - 27,poleKategorii[j]
                            .poleOdkazu[k].napis.getBounds().width,poleKategorii[j]
                            .poleOdkazu[k].napis.getBounds().height);
                }
            }
```

Obrázek 25: Ukázka části algoritmu starajícího se o sbalení kategorie

Dále nezávisle na tom, zda byla kategorie sbalena nebo rozbalena, je potřeba nastavit novou preferovanou velikost panelu, podle které se řídí zobrazovaní posuvníků.

9.4 Stránky s teorií

Všechny následující kódy, které budou popsány v této kapitole, se nacházejí ve třídě s názvem „PdfStranka“, která představuje stránky s teorií.

Stránky s teorií jsou vytvořeny v Microsoft Wordu a dále převedeny do PDF formátu. Vzhledem k tomu, že Java bez implementace různých appletů nepodporuje PDF formát, jsou stránky převedeny do JPG formátu a následně vloženy do programu. Ale jak už bylo zmíněno v předchozích kapitolách, stránky v PDF formátu jsou také použity, ale mimo program, a dají se pomocí programu externě spustit v PDF prohlížeči.

Konstruktor této třídy má jediný parametr, a to právě adresu k určenému obrázku. Tato adresa je dále v konstruktoru přetransformována tak, aby odpovídala adrese k odpovídajícímu PDF souboru (viz Obrázek 26). A tato přetransformovaná adresa je dále použita mimo třídu pro externí otevření tohoto PDF souboru.

```
//Upravení adresy souboru jpg na adresu pdf souboru jemu odpovidajicimu.
adresaPDF = soubor.replaceFirst("/pdfobrazky/","").replaceAll("jpg", "pdf");
```

Obrázek 26: Transformace adresy k JPG souboru na adresu k PDF souboru.

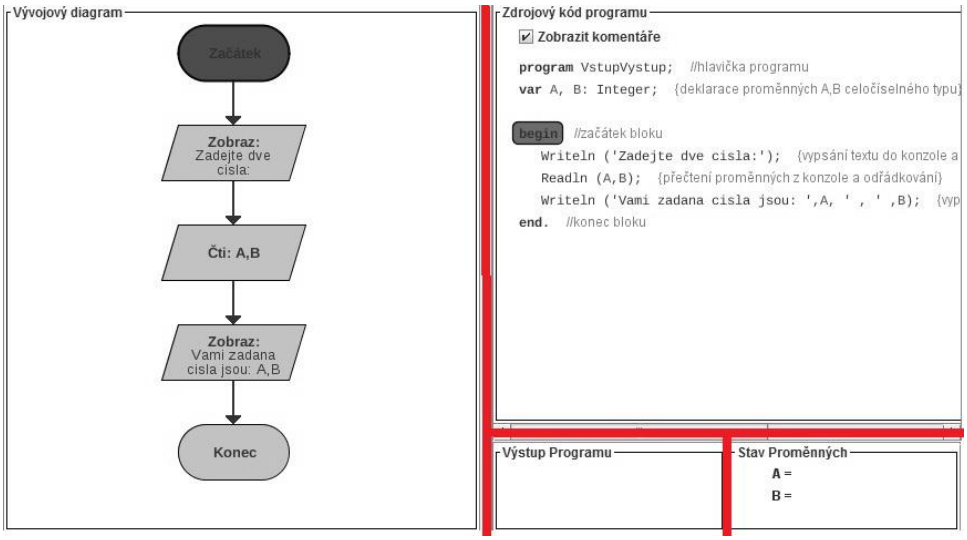
Stránka se ihned po zobrazení stane vlastníkem zaměření, tudíž jí je jako všem ostatním stránkám přidělen posluchač stisknutí kláves s názvem „TlacitkoKeyListener“.

9.5 Stránky s příklady

Zatímco pro všechny ostatní stránky stačila jedna třída, pro stránky s příklady je použito tříd rovnou několik.

9.5.1 Vytvoření panelů

Vzhledem k tomu, že stránky s příklady jsou velice členěné (viz Obrázek 27), je potřeba nejdříve jednotlivé rozdělení panelů připravit.



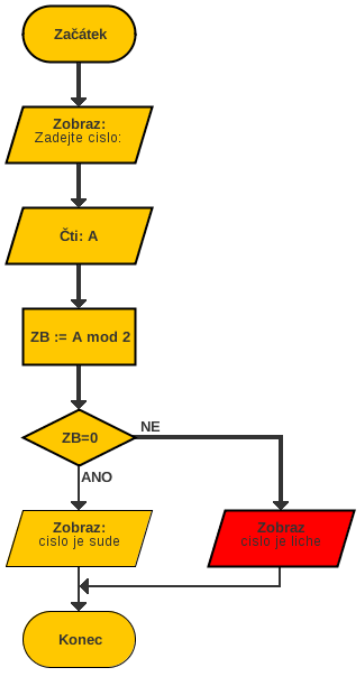
Obrázek 27: Ukázka rozdělení panelů

Horizontální rozdělení okna na dvě půlky je zajištěno třídou s názvem „DvojOkno“, která představuje dva panely, které jsou odděleny posuvnou lištou. Zatímco v levém panelu se již nachází diagram, pravý panel je ještě potřeba rozdělit na další části. To je zajištěno další třídou s názvem „PravyPanel“, ve které je rozdělen pravý panel nejdříve na horní panel se zdrojovým kódem a na panel dolní. Přičemž dolní panel je dále rozdělen na dva panely pro výstup programu a stav proměnných. Ve všech těchto panelech se nacházejí posuvné lišty, které se zobrazí v případě potřeby.

Jak již bylo zmíněno, každá stránka má svůj posluchač přidělení zaměření a stejně tomu je i u stránky s příklady. Zde se ale kromě vypnutí a zapnutí určitých tlačítek na dolním panelu objevuje i předání zaměření prvnímu objektu v panelu, jehož konkrétní využití bude popsáno v dalších kapitolách.

9.5.2 Abstraktní třídy pro příklad

Nyní bude popsána nejdůležitější část programu. Tou jsou 3 abstraktní třídy, které slouží pouze jako abstraktní rodičovské třídy pro konkrétní třídy představující již jednotlivé příklady v programu. Tyto abstraktní třídy jsou „DiagramPanel“, která představuje panel s vývojovým diagramem (viz Obrázek 28). Tato třída je nejrozsáhlejší a je v ní většina algoritmu zodpovídajícího za interaktivnost příkladu. Druhou třídou je „ZdrojovyKodPanel“, která představuje panel se zdrojovým kódem. Třetí třídou je „PromennePanel“, která představuje panel se zobrazenými proměnnými.



Obrázek 28: Ukázka vývojového diagramu

Třída „DiagramPanel“

Pro veškerý text v diagramu byl použit open-source font jménem „Liberation sans“. Interní font byl zvolen z toho důvodu, aby byl program plně multiplatformní. Java jinak totiž používá font z operačního systému, na kterém je program spuštěn, a tudíž může mít na každém operačním systému tento font jiné rozměry. V některých systémech by se tedy text například nemusel vejít do objektu.

Nyní bude vysvětleno, k čemu je primárně v programu použito zaměření, tedy kromě své samotné funkce, kterou je přijímání vstupu z klávesnice vlastníkem zaměření. Využije se toho, že pro zaměření existuje posluchač, který dokáže indikovat, kdy bylo zaměření objektu přiděleno a kdy bylo odebráno. Další výhodou zaměření je možnost nastavit tzv. okruh zaměření (focus okruh), který zajistí, že všechny zaměřitelné objekty v určeném komponentu budou zařazeny do řady. Pro tento okruh existují funkce, které posunou zaměření na další nebo předchozí objekt

v řadě. Pro panel s diagramem je tedy nastaven takovýto okruh. Dále bude popsáno, jak bylo využito dalších výhod zaměření.

Všechny grafické objekty v diagramu jsou vykreslovány přímo programem. Každý grafický objekt má ve třídě „DiagramPanel“ podtřídu, která má za úkol vykreslit právě tento objekt.

Nejdůležitější jsou v diagramu objekty představující nějaký krok algoritmu (ovál, obdélník, kosodélník, atd.). Podtřídy starající se o vykreslení těchto objektů jsou potomkem jedné společné abstraktní třídy s názvem „Objekt“, která se stará o většinu interaktivních akcí v diagramu.

Podtřída „Objekt“

Nápis v objektu je umístěn do panelu jako samostatný prvek, nicméně jeho hranice a umístění se při každé změně přepočítávají tak, aby odpovídaly hranicím a umístění objektu, pod který spadá.

Důležité pro používání zaměření v diagramu je nastavení zaměřitelnosti objektů, protože grafické objekty nemají důvod být zaměřitelné, a tudíž defaultně nejsou.

Aby zaměření u objektu plnilo svou funkci, je potřeba objektu také přidělit posluchač zaměření. V případě, že objekt získá zaměření, stane se označeným objektem (v diagramu červeně označeným). Při označení objektu se musí označit i příslušný řádek ve zdrojovém kódu. V případě, že se jedná o objekt, který představuje krok pro zadání vstupních dat, musí být při označení vyvolán dialog pro zadání vstupních dat. Pro vyvolání dialogu je ve třídě „Objekt“ podtřída „Dialog“, která bude popsána dále.

V případě ztráty zaměření stačí postupovat v podstatě opačným způsobem, tedy nastavit objektu původní barvu a řádku zrušit pozadí.

Vzhledem k tomu, že vlastníkem zaměření u ostatních stránek byly samotné stránky, tak i ony samotné měly přidělen posluchač stisknutí kláves na klávesnici. Nicméně v tomto případě bude vlastníkem zaměření vždy jeden z objektů, a tudíž musí mít každý z nich přidělen již zmiňovaný posluchač s názvem „TlacitkoKeyListener“.

Posledním posluchačem, který je přidělen těmto objektům, je posluchač myši s názvem „DiagramMouseListener“. Tento posluchač je ale přidělován objektům jenom v případě, že se nejedná o příklad s cyklem, protože v příkladech s cyklem není dovoleno klikat na objekty. V případě, že bylo na objekt kliknuto levým tlačítkem myši, se tomuto objektu přidělí zaměření. Pokud bylo na objekt pouze najeto kurzorem, zvýrazní se okraje objektu.

Objekty jsou po vytvoření vloženy do pole, které představuje všechny označitelné objekty v panelu.

V parametru konstruktoru třídy „Objekt“ byl zadán příslušný řádek tomuto objektu. Takže objekt již má informaci o tom, jaký řádek v panelu se zdrojovým kódem k němu patří. Ale řádek nemá žádnou informaci ohledně toho, který objekt v diagramu k němu patří. Je to z toho důvodu, že panel se zdrojovým kódem je vytvořen dříve než panel s diagramem, a tudíž není možné při vytváření řádku ve zdrojovém kódu zadat, že k němu patří určitý objekt z diagramu, když objekt ještě v danou chvíli neexistuje. Nicméně, když je vytvořen objekt v digramu, byl již vytvořen i řádek ve zdrojovém kódu, tudíž objektu je možné předat informaci, který k němu patří řádek. Ve chvíli, kdy je vytvořen objekt, je možné zpátky příslušnému řádku předat informaci, jaký k němu patří objekt.

Pro nastavení správného průchodu diagramem jsou zapotřebí dvě metody s názvem „nastavitNeoznacitelny“ a „nastavitOznacitelny“. Označitelné smějí být pouze ty objekty, které zrovna odpovídají současnému stavu vstupních hodnot, aby uživatel nemohl kliknout na objekt, který neodpovídá stavu proměnných. Metoda „nastavitNeoznacitelny“ nastaví objektu nemožnost získat zaměření, odebere mu posluchač myši a příslušnému řádku nastaví podobnou metodou to samé. V případě metody „nastavitOznacitelny“ se provede opak.

Tyto dvě metody jsou používány převážně uvnitř metody s názvem „zablokovatObjekty“, což je abstraktní metoda umístěná ve třídě „DiagramPanel“, která má pro každý příklad jinou implementaci. Respektive se v ní definuje algoritmus programu, který představuje samotný příklad.

Třída „Objekt“ obsahuje velké množství metod pro vykreslení šipek od přímých až po různě zahnuté. Tyto metody využívají obecné třídy na vytváření šipek, které jsou umístěny mimo třídu „Objekt“. Každá šipka nese informaci o tom, u kterého objektu začíná a u kterého končí.

Metoda „Oznacit“

Nyní bude popsána rozsáhlá metoda „oznacit“, která zdaleka nemá za úkol pouze zvýraznit objekt červenou barvou. Vzhledem k tomu, že na jednotlivé objekty je možné klikat a uživatel tím pádem může přeskočit na libovolný krok, je potřeba vždy přehodnotit celý diagram znovu a nevyhodnocovat pouze další krok.

Nejdříve tedy přichází fáze, kdy je zapotřebí všechno uvést do počátečního stavu. Vymaže se panel s výstupem, vymažou se hodnoty proměnných v panelu s proměnnými. Vymažou se všechny šipky, které byly vykresleny tučně. U všech objektů se zruší tučný okraj.

Ve druhé fázi je uveden diagram do stavu, v jakém má aktuálně být. To se provede pomocí cyklu, který prochází pole objektů od prvního až po současně označený. Při procházení pole objektů jsou důležité pouze ty objekty, které jsou označitelné. Začíná se tím, že se u všech označitelných objektů nastaví tučné okraje.

Nyní je potřeba vykreslit tučné šipky, které překryjí šipky normální, a vznikne tím dojem ztučnění šipek. Nicméně tento krok je mnohem náročnější než předchozí. Algoritmus ztučňování šipek probíhá tak, že se nejdříve zjistí, je-li současný objekt označitelný a zda je označitelný i předchozí objekt (viz Obrázek 29). V případě, že obojí platí, stačí zjistit, který druh šipky má být vykreslen. Pokud je současný objekt označitelný a předchozí ne, musí se postupně pomocí cyklu prohledat pole ještě hlouběji, dokud se nenarazí na označitelný objekt, ze kterého vede šipka k současně označenému (viz Obrázek 29).

```
if (poleObjektu[i].isFocusable()){
    if (poleObjektu[i-1].isFocusable()){
        if(poleObjektu[i-1].sipka != null && (poleObjektu[i-1].sipka.equals(poleObjektu[i]))){
            levyPanel.add(getSipkaDolu(poleObjektu[i-1],poleObjektu[i-1].sipka,true));
        }else if(poleObjektu[i-1].sipkaZBoku != null){
            poleObjektu[i-1].vykreslitSipkuZBokuTucnou();
        }else if(poleObjektu[i-1].sipkaDoBoku != null){
            for(int m=0;m<poleObjektu[i-1].sipkaDoBoku.length;m++){
                if(poleObjektu[i-1].sipkaDoBoku[m].zakonceni.isFocusable()){
                    poleObjektu[i-1].vykreslitSipkuDoBokuTucnou(poleObjektu[i-1].sipkaDoBoku[m]);
                }
            }
        }
    }else{
        for(int l = i-1;l>0;l--){
            if(poleObjektu[l].isFocusable()){
                if(poleObjektu[l].sipka != null && poleObjektu[l].sipka.equals(poleObjektu[i])){
                    levyPanel.add(getSipkaDolu(poleObjektu[l],poleObjektu[l].sipka,true));
                }else if(poleObjektu[l].sipkaZBoku != null){
                    poleObjektu[l].vykreslitSipkuZBokuTucnou();
                }
                break;
            }
        }
    }
}
```

Obrázek 29: Ukázka části algoritmu starajícího se o zvýraznění šipek

Dále je potřeba zobrazit hodnoty proměnných v panelu s proměnnými. Využívá se zde řádků ze zdrojového kódu příslušných právě procházeným objektům, tedy konkrétně jejich textu, který představuje příkazy z jazyka Pascal, takže je využito sémantiky tohoto jazyka. Konkrétně je vyhledáván příkaz pro čtení, tedy řetězec „Read“, a příkazy pro přiřazování, tedy řetězec „:=“. V případě, že řádek odpovídající tomuto objektu obsahuje řetězec „Read“, je třeba v panelu s proměnnými změnit hodnoty proměnných odpovídající vstupním hodnotám. V případě, že příslušný řádek obsahuje řetězec „:=“, musí se hodnoty nejdříve vypočítat. Výpočet hodnot probíhá v metodě s názvem „vypocitat“ a jedná se o druhou abstraktní metodu z třídy „DiagramPanel“. Tato metoda je implementována pro každý příklad zvlášť. Hodnoty se v ní vypočítají a zároveň zobrazí v panelu s proměnnými.

Nyní zbývá poslední krok při označení objektu, a to zobrazení výstupu programu v panelu s výstupem. Zobrazení probíhá plně automaticky pro všechny příklady bez individuální

implementace pro jednotlivé příklady. Využije se zde zase textu z příslušných řádků a tentokrát je vyhledáván řetězec „Writeln“ nebo „Write“. Samotný text, který má být vypsán v panelu včetně proměnných, je získán postupným odebíráním částí textu zleva doprava, přičemž nechtěné části textu (sémantika jazyka Pascal) jsou odstraňovány a chtěné části textu (řetězec a proměnné, které mají být vypsány) jsou ukládány do proměnné, která ve výsledku představuje kompletní text pro vypsání v panelu pro výstup (viz Obrázek 30). Hodnotu proměnné, která má být vypsána v určitém řádku, má každý řádek uloženu v sobě. Nakonec stačí kompletní text pouze zobrazit v panelu pro výstup metodou „pridatRadek“.

```
//Vypsání obsahu řádku.
for(int p=0;p<textKVypsani.length();p++){
        if(textKVypsani.startsWith("Writeln (")){
            textKVypsani = textKVypsani.replace("Writeln (","");
        }else if(textKVypsani.contains("Writeln (")){
            textKVypsani = textKVypsani.substring(textKVypsani.indexOf("Writeln ("));
            textKVypsani = textKVypsani.replace("Writeln (","");
        }
        if(textKVypsani.startsWith("'")){
            castTextu = textKVypsani.substring(textKVypsani.indexOf("'")+1);
            textKVypsani = textKVypsani.replaceFirst("'","");
            castTextu = castTextu.substring(0,castTextu.indexOf("'"));
            textKVypsani = textKVypsani.replaceFirst(castTextu+"'","");
        }else if(textKVypsani.startsWith(" '") ){
            castTextu = textKVypsani.substring(textKVypsani.indexOf("'")+1);
            textKVypsani = textKVypsani.replaceFirst(" '","");
            castTextu = castTextu.substring(0,castTextu.indexOf("'"));
            textKVypsani = textKVypsani.replaceFirst(castTextu+"'","");
        }else if(textKVypsani.startsWith(",")){
            pomocnaCastTextu = textKVypsani.substring(textKVypsani.indexOf(",")+1);
            textKVypsani = textKVypsani.replaceFirst(",","");
            if(pomocnaCastTextu.contains(",")){
                pomocnaCastTextu = pomocnaCastTextu.substring(0,pomocnaCastTextu.indexOf(","));
                textKVypsani = textKVypsani.replace(pomocnaCastTextu+",","");
            }else{
                pomocnaCastTextu = pomocnaCastTextu.substring(0,pomocnaCastTextu.indexOf(")"));
                textKVypsani = textKVypsani.replace(pomocnaCastTextu,"");
            }
```

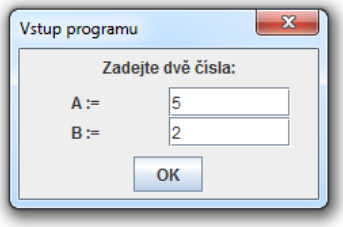
Obrázek 30: Ukázka části algoritmu starajícího se o vyseparování řetězce z příkazu

Metoda „posunout“

Metoda „posunout“ se nachází ve třídě „Objekt“ a má za úkol automaticky posouvat posuvníky. Snahou bylo, aby automatické posouvání lišt bylo co nejpřirozenější, a proto jsou zde dva druhy posunutí (pokud nepočítáme směr). Pokud je právě označený objekt celý viditelný, posuvníky se posunou o kousek dolu nebo nahoru při stisknutí tlačítka se šipkou. Ale pokud právě označený objekt již není vidět vůbec nebo jen z části, posuvníky se posunout o nejmenší potřebnou vzdálenost tak, aby byl objekt vidět celý. Horizontální posuvníky se posouvají automaticky pouze v případě, že objekt nebyl vidět. Úplně stejně probíhá posouvání na panelu se zdrojovým kódem, které je potřeba provádět zároveň. Směr posunutí se nastavuje podle stisknutého tlačítka na dolním panelu.

Podtřída „Dialog“

Dále třída „Objekt“ obsahuje podtřídu s názvem „Dialog“, která je zodpovědná za nastavení a zobrazení dialogu, ve kterém se zadávají vstupní hodnoty příkladu (viz Obrázek 31). Dialog má pro každý příklad jiné rozměry, jiný počet polí pro zadávání proměnných, jiný nadpis a hlavně jiný datový typ zadávaných hodnot. Konstruktor této třídy má zajistit dynamické vytvoření konkrétního dialogu.



Obrázek 31: Ukázka dialogu pro zadání vstupních hodnot

Svou velikost dialog upravuje podle počtu textových polí pro zadání hodnot. Tato velikost je neměnná, tedy uživatel nemůže okno zvětšovat ani zmenšovat.

Dialogu je znemožněno vypnutí křížkem, takže je možné ho vypnout pouze stisknutím tlačítka „OK“ a tedy potvrzením zadaných hodnot. Navíc je dialog modální, což znamená, že není umožněno cokoli provádět v aplikaci, která ho vyvolala, dokud ho uživatel nepotvrdí.

Pro textová pole v dialogu je nastaven povolený formát jejich obsahu podle datového typu hodnot, které se do nich mají zadávat. Nejjednodušší je to v případě řetězce, protože zadávání řetězce do textového pole je standardní. V případě znaků je pouze potřeba omezit maskou řetězec na pouhý jeden znak.

Pro datový typ Integer je využito výhody, že Java má stejný rozsah datového typu Integer jako překladač Turbo Pascal. Je tedy možné textovému poli nastavit formát Integer, který v případě překročení rozsahu způsobí výjimku.

Při vytváření dialogu je zadána defaultní hodnota každého textového pole, která je zde hlavně z toho důvodu, aby uživatel mohl kliknout v diagramu na jakýkoli krok a přeskočit tak zadávání vstupních hodnot a právě v tomto případě jsou použity defaultní hodnoty. Defaultní hodnoty se také zobrazí při prvním zobrazení dialogu.

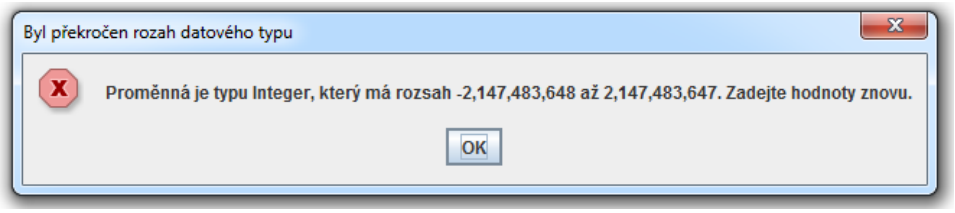
Textovým polím je přidělen posluchač stisknutí kláves na klávesnici, který pouze v případě stisknutí klávesy enter, přesune zaměření na další zaměřitelný objekt v dialogu. Textová pole a tlačítka jsou defaultně zaměřitelná, tudíž při zobrazení dialogu dostane zaměření automaticky první textové pole.

Tlačítku „OK“ je přidělen posluchač získání zaměření a v případě, že poslední textové pole předalo zaměření tomuto tlačítku, tlačítko se automaticky stiskne, aby se dialog ukončil.

Tlačítko „OK“ má přidělen ještě jeden posluchač právě pro stisknutí tlačítka. V případě, že je tlačítko stisknuto, dialog se ukončí (pouze zneviditelní).

Posledním a nejdůležitějším posluchačem v dialogu je posluchač s názvem „DialogPropertyChangeListener“, který je přidělen všem textovým polím a jeho metoda „propertyChange“ se vykoná v případě, že byla změněna hodnota v textovém poli.

V metodě „propertyChange“ je nejdříve zjištěno, které textové pole vyvolalo tuto událost. Poté je zjištěno, jakého datového typu je textové pole. V případě znaků a řetězců je další postup jednoduchý, protože řetězce nemají žádné omezení a v případě znaků uživateli není dovoleno napsat špatnou hodnotu (víc jak jeden znak). Stačí tedy pouze přečíst hodnotu z textového pole a zobrazit ji jako hodnotu příslušné proměnné v panelu s proměnnými. Složitější je to v případě zadávání čísel, protože tam je povoleno zadat hodnotu přesahující rozsah datového typu Integer. Nicméně protože byl pro textové pole nastaven formát Integer, je v případě zadání špatné hodnoty (přesahující rozsah) vyvolána výjimka, která je zároveň odchycena. V odchycené výjimce je zobrazen nový dialog, který upozorňuje uživatele na překročení rozsahu datového typu Integer a vyzve ho k opětovnému zadání nových hodnot (viz Obrázek 32).



Obrázek 32: Chybový dialog zobrazený při překročení rozsahu Integer

Nyní se vrátíme k předchozí třídě „DiagramPanel“, ve které je několik podtříd, které jsou potomkem právě popsané třídy „Objekt“. Tyto třídy představují označitelné objekty v diagramu (ovál, obdélník, kosodélník, atd.) a obsahují jedinou metodu „paint“, která má za úkol vykreslit objekt. Právě v této metodě je definován tvar, výplň a další grafické prvky objektu.

Dále je zde podtřída s názvem „Napis“, která se stará o vykreslení jednoho řádku textu v diagramu. Tato třída je použita i ve třídě „Objekt“ pro vykreslení primárního nápisu uvnitř objektu. Opět se zde nachází metoda „paint“, která je vykonána při vykreslování nápisu. Je v ní nastaven font a hlavně umístění nápisu uvnitř objektu. Aby byl text vystředěn uvnitř objektu, je potřeba zjistit i jeho délku v pixelech a zahrnout ji do výpočtu umístění.

Třída „DiagramPanel“ obsahuje také velké množství tříd pro vykreslení různých druhů šipek. Jejich konstruktory jsou velice podobné a zpravidla obsahují tyto parametry: objekt, od kterého má šipka vést, objekt, ke kterému má šipka vést, horizontální a vertikální prodloužení

šipky. Jinak obsahují tyto třídy opět metodu „paint“ ve které je pomocí čar sestavena šipka (viz Obrázek 33).

```
@Override
public void paint(Graphics g){
  //Přetypování obyčejné grafiky na 2D
  Graphics2D g2 = (Graphics2D)g;

  //Nastavení tloušťky čáry.
  if (tlustsi){
      g2.setStroke(new BasicStroke(4));
  }else{
      g2.setStroke(new BasicStroke(2));
  }

  //Vykreslení cesty šipky.
  GeneralPath sipka = new GeneralPath();
  sipka.moveTo(umisteni.getLocation().x+102,umisteni.getLocation().y + 27);
  sipka.lineTo(zakonceni.getLocation().x+(zakonceni.getWidth()/2)
          +prodlouzeniX,umisteni.getLocation().y+27);
  sipka.lineTo(zakonceni.getLocation().x+(zakonceni.getWidth()/2)
          +prodlouzeniX,zakonceni.getLocation().y+1+prodlouzeniY-8);

  //Vykreslení kompletní šipky.
  g2.draw(sipka);

  //Vykreslení konce šipky.
  vykresleniKonceSipky(g2, SwingConstants.SOUTH,zakonceni.getLocation().x
          +(zakonceni.getWidth()/2)+prodlouzeniX,zakonceni.getLocation().y+1+prodlouzeniY);
}
```

Obrázek 33: Ukázka metody pro vykreslení šipky do boku

Třída „ZdrojovyKodPanel“

Tato třída má na starost panel se zdrojovým kódem (viz Obrázek 34) a funguje velice podobně jako předchozí třída „DiagramPanel“. Většina operací mezi těmito třídami probíhá křížem, tedy pokud se něco stane ve zdrojovém kódu, musí se i něco stát v diagramu a opačně.

☑ Zobrazit komentáře

```
program DeleniDvouCisel;
var A, B, Cela_cast, Zbytek: Integer;
    Podil: Real;

begin
   Writeln ('Zadejte dve cisla:');
   Readln (A,B);
   if B=0 then  {zjištění, zda jmenovatel není nulový}
      Writeln ('Nelze delit nulou!')  {vypíšeme chybovou hlášku v případě, že B=0}
   else
      begin  {provedeme delení a vypíšeme výsledky dělení v přípůadě, že B se nerovná 0}
         Cela_cast := A div B;  {celočíselné dělení}
         Zbytek := A mod B;  {zbytek po celočíselném dělení}
         Podil := A/B;  {reálný podíl}
         Writeln ('Celociselny podil: ',Cela_cast);
         Writeln ('Zbytek po deleni: ',Zbytek);
         Writeln ('Realny podil: ',Podil);
      end;
end.
```

Obrázek 34: Ukázka panelu se zdrojovým kódem

Pro text je zde opět použit open-source font jménem „Liberation Mono“, který je vhodný pro zobrazování zdrojového kódu.

Podtřída „Text“

Nejdříve bude popsána podtřída „Text“, která představuje jeden řádek textu ve zdrojovém kódu. V případě, že bylo parametrem zadáno, že tento řádek má být označitelný, musí být nastaven jako zaměřitelný. Dále musí mít označitelný řádek přidělen posluchač získání zaměření, posluchač myši a posluchač stisknutí kláves na klávesnici. Zároveň má každý označitelný řádek přiděleno pozadí ve tvaru oválu, které je zobrazeno při najetí kurzoru myši na řádek nebo při označení řádku.

Dále třída „Text“ obsahuje metodu pro automatické zmodrání textu (viz Obrázek 35), který se nachází mezi uvozovkami. Tato metoda je vykonána při vytváření každého řádku.

```
//Zmodrání textu.
private void automatickyZmodrat(){
    if(text.indexOf("'")>=0){
        zbarvenyText.addAttribute(TextAttribute.FOREGROUND,new Color(4,51,162),text.indexOf("'"),
                text.indexOf("'",text.indexOf("'")+1)+1);
        zbarvenyText.addAttribute(TextAttribute.FONT,font,text.indexOf("'"),
                text.indexOf("'",text.indexOf("'")+1)+1);
        if(text.indexOf("'",text.indexOf("'",text.indexOf("'")+1)+1)>=0){
            zbarvenyText.addAttribute(TextAttribute.FOREGROUND,new Color(4,51,162),
                    text.indexOf("'",(text.indexOf("'",text.indexOf("'")+1))+1),
                    text.indexOf("'",(text.indexOf("'",(text.indexOf("'",text.indexOf("'")+1))+1))+1)+1);
            zbarvenyText.addAttribute(TextAttribute.FONT,font,text.indexOf("'",(text.indexOf("'",
                    text.indexOf("'")+1))+1),text.indexOf("'",(text.indexOf("'",
                    (text.indexOf("'",text.indexOf("'")+1))+1))+1)+1);
        }
    }
}
```

Obrázek 35: Metoda pro zmodrání textu nacházejícího se mezi uvozovkami

Podobnou metodou je metoda na ztučnění určitých slov v textu (klíčových slov), jíž je jako parametr předáno právě slovo, které má být v textu ztučněno. Tato metoda je vykonána při vytváření každého řádku a zároveň pro každé klíčové slovo.

Dále je zde metoda pro zezelenání všech identifikátorů proměnných. Této metodě je zadán jako parametr právě identifikátor, který má být v celém zdrojovém kódu zeleně vyznačen.

Panel se zdrojovým kódem má opět nastaven vlastní okruh zaměření a chová se obdobně jako panel s diagramem. Každému označitelnému řádku je přidělen posluchač zaměření, který funguje také obdobně jako pro objekty v diagramu. Uživatel v podstatě nemá šanci poznat, kde se zrovna zaměření nachází, jestli na objektu v diagramu nebo na řádku ve zdrojovém kódu, protože krokování probíhá paralelně v obou panelech.

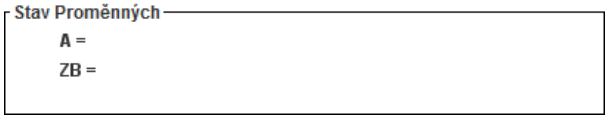
Přesto, že obě třídy se chovají velice podobně, je většina algoritmu přeci jen ponechána na panelu s diagramem, protože například panel se zdrojovým kódem se vůbec nestará o zobrazování proměnných nebo výstupu programu.

Dále se ve zdrojovém kódu zobrazují komentáře, které nemají žádné interaktivní funkce, pouze se dají vypnout a zapnout. Pro jejich vypnutí a zapnutí slouží zaškrtávací tlačítko (viz Obrázek 34) na vrcholu panelu, které pouze zobrazuje nebo odstraňuje komentáře.

Třída „PromennePanel“

Pro každý datový typ použitý v programu je vytvořeno pole proměnných o velikosti dané celkovým počtem proměnných. Zároveň je vytvořeno pole nápisů představující kompletní řádek vypsaný v panelu, tedy jednotlivé identifikátory proměnných a jejich hodnoty.

První metodou v této třídě je metoda „pridatPromennou“, která má jediný parametr, v němž je jí zadán statický text před hodnotou proměnné (viz Obrázek 36). V této metodě je nápis vytvořen a umístěn do panelu.



Obrázek 36: Ukázka panelu s proměnnými

Podtřída „NapisPromenne“

Dále se zde nachází podtřída „NapisPromenne“, která představuje jeden nápis včetně hodnot proměnných. Parametrem konstruktoru je jí nejdříve zadán statický text před hodnotou a samotná hodnota se již přidává a upravuje metodami. Tyto metody se liší pouze datovým typem parametru, který představuje právě hodnotu proměnné (viz Obrázek 37).

```
//Metoda pro aktualizaci hodnoty proměnné.
void zmenitNapis(int promenna){
    text = textNaZacatku + promenna;
    zjisteniNejdelsihoTextu();
}
```

Obrázek 37: Metoda pro změnu hodnoty proměnné datového typu Integer

Další metoda v třídě „NapisPromenne“ je „zjisteniNejdelsihoTextu“, která má za úkol zjistit, zda je tento text nejdelší v panelu a v případě, že je, podle něj nastavit preferovanou velikost panelu, podle které se řídí zobrazování posuvníků.

Nyní se vrátíme k původní třídě „PromennePanel“, kde se nachází několik metod pro změnu hodnoty jedné proměnné. Tyto metody se opět liší pouze v datovém typu svého parametru. Je v ní tedy změněn celkový text nápisu tak, aby obsahoval i hodnotu nebo pouze aktualizoval hodnotu. Dále je hodnota proměnné uložena do pole s proměnnými, které je použito v celém programu pro zjištění hodnot proměnných. Je zde také provedena metoda „navazatNaRadek“.

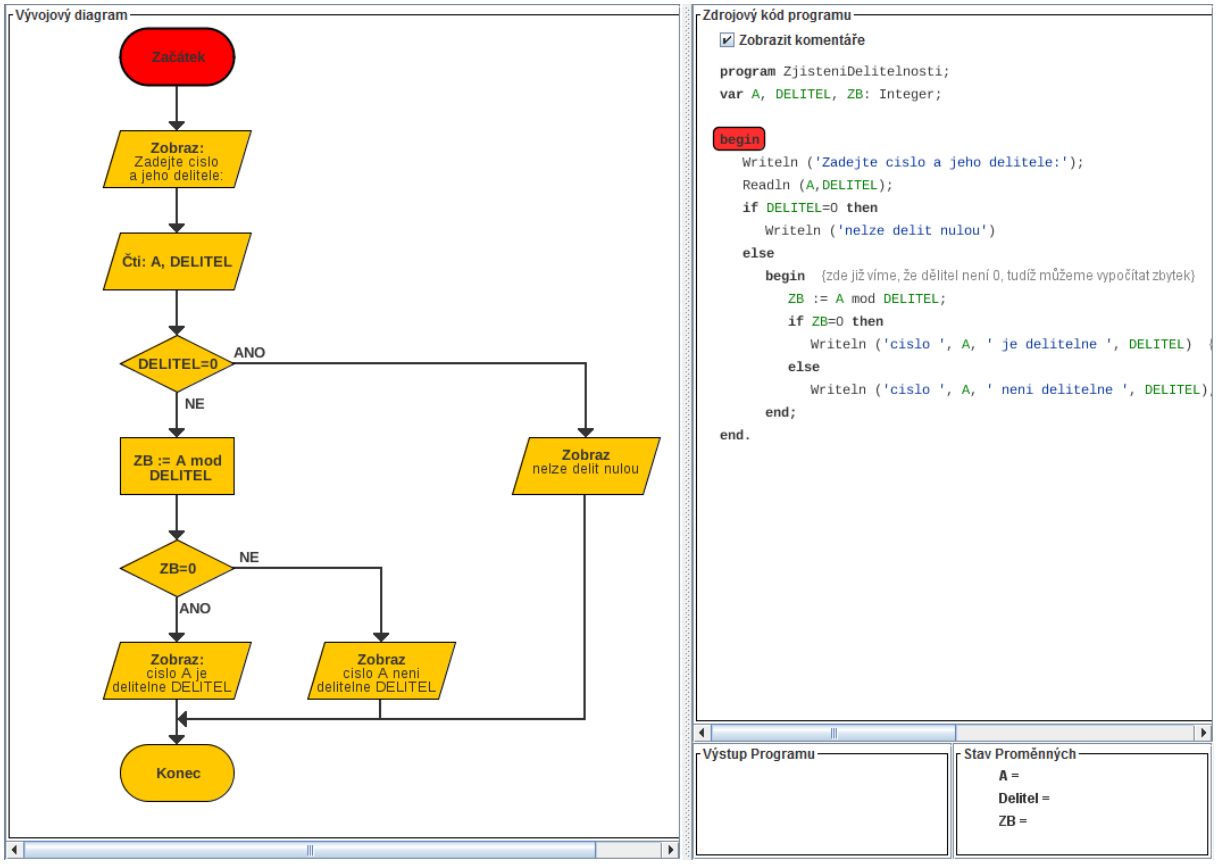
Metoda „navazatNaRadek“ je abstraktní a definuje se při vytváření konkrétního příkladu. Pokud má být některá proměnná vypsána ve výstupu, tak právě v této metodě je definováno, ke kterému řádku tato proměnná patří, aby mohla být později vypsána. Tato metoda se provede při každé změně hodnot proměnných, aby byly hodnoty stále aktuální.

9.5.3 Třída „PanelVystup“

Tato třída je společná pro všechny příklady a představuje panel pro zobrazení výstupu programu. Jak už bylo popsáno, vypisování výstupu probíhá zcela automaticky, proto není potřeba tuto třídu zvlášť implementovat pro každý příklad. V této třídě jsou pouze metody pro automatické zobrazení výstupu.

9.5.4 Vytvoření příkladu

Pro vytvoření jednoho konkrétního příkladu je potřeba vytvořit tři nové třídy, které budou potomky abstraktních tříd z minulé kapitoly. Tyto tři třídy se jmenují „DiagramStranka“, „ZdrojovyKodStranka“ a „PromenneStranka“. V následujících kapitolách bude vysvětleno obecné vytvoření příkladu. Konkrétní ukázky kódu nebo stránky budou z příkladu pro zjištění dělitelnosti jednoho čísla druhým (viz Obrázek 38).



Obrázek 38: Ukázka stránky s příkladem (Dělitelnost jednoho čísla druhým)

Třída „DiagramStranka“

Nejdříve se musí vytvořit označitelné objekty v diagramu (viz Obrázek 39). U každého objektu se pomocí parametrů nejprve zadává jeho umístění, přičemž je dodržováno jakési tabulkové rozložení objektů v panelu, tedy jsou dodržovány stále stejné vzdálenosti mezi objekty. Jako další parametr je uveden primární text v objektu. Dalším nepovinným parametrem je objekt, u kterého bude končit šipka dolu z tohoto objektu. Dva předposlední parametry jsou většinou nevyužity, protože se starají o případné dorovnání textu v objektu. A jako poslední parametr se udává řádek ze zdrojového kódu, který odpovídá tomuto objektu.

```
//Vytvoření objektů v diagramu.
konec = new Oval(20,650,"Konec",0,0,ZdrojovyKodStranka10.radek16);
zobrazeni4 = new Rovnobeznik(380,380,"Zobraz",0,-10,ZdrojovyKodStranka10.radek7);
zobrazeni3 = new Rovnobeznik(200,560,"Zobraz",0,-10,ZdrojovyKodStranka10.radek14);
zobrazeni2 = new Rovnobeznik(20,560,"Zobraz:",konec,0,-10,ZdrojovyKodStranka10.radek12);
podminka2 = new Kosodelnik(20,470,"ZB=0",zobrazeni2,0,0,ZdrojovyKodStranka10.radek11);
vypocet = new Obdelnik(20,380,"ZB := A mod",podminka2,0,-5,ZdrojovyKodStranka10.radek10);
podminka1 = new Kosodelnik(20,290,"DELITEL=0",vypocet,0,0,ZdrojovyKodStranka10.radek6);
vstup = new Rovnobeznik(20,200,"Čti: A, DELITEL",podminka1,0,0,ZdrojovyKodStranka10.radek5);
zobrazeni1 = new Rovnobeznik(20,110,"Zobraz:",vstup,0,-10,ZdrojovyKodStranka10.radek4);
zacatek = new Oval(20,20,"Začátek",zobrazeni1,0,0,ZdrojovyKodStranka10.radek3);
```

Obrázek 39: Ukázka vytvoření objektů v diagramu

Nyní, když jsou objekty vytvořeny, je možné je nahrát do pole, které určuje, jaký objekt mění které proměnné (viz Obrázek 40). Toto pole má dva rozměry; první udává, o kolikátou proměnnou se jedná, a druhý rozměr udává pořadí objektu, který smí měnit proměnnou.

```
//Naplnění pole objektama které mění proměnné v programu.
poleObjektuSPromennyma = new Objekt[3][1];
poleObjektuSPromennyma[0][0] = vstup;
poleObjektuSPromennyma[1][0] = vstup;
poleObjektuSPromennyma[2][0] = vypocet;
```

Obrázek 40: Naplnění pole udávající, který objekt smí měnit kterou proměnnou

Jako další je potřeba nastavit dialog pro zadávání vstupních hodnot. Dialog je vytvořen v objektu, který ho má vyvolávat, a jako parametr je mu zadán nadpis dialogu a počet vstupních hodnot (viz Obrázek 41). Poté se musí do dialogu přidat příslušné textové pole metodou „pridatPole“, které se jako parametr zadávají: pořadí pole, text před polem a defaultní hodnota pole (viz Obrázek 41).

```
//Nastavení dialogu
dialog = vstup.getDialog("Zadejte číslo a dělitele:",2);

//Naplnění dialogu.
dialog.pridatPole(1,"         A :=",8);
dialog.pridatPole(2,"   Delitel :=",2);
```

Obrázek 41: Vytvoření dialogu pro zadávání vstupu

V dalším kroku musí být zadána adresa k odpovídajícímu PDF souboru se zadáním příkladu.

Dále se musí uvést všechny sekundární nápisy v objektech (viz Obrázek 42). Sekundárním nápisem je myšlen druhý a každý další řádek v objektu. V parametrech pro sekundární nápis jsou uvedeny: samotný text, objekt, v kterém má být text vytvořen, umístění textu v objektu a pravdivostní proměnná určující tučnost písma.

```
//Nastavení sekundárních nápisů
levyPanel.add(getNapis("Zadejte cislo",zobrazeni1,0,2,true));
levyPanel.add(getNapis("a jeho delitele:",zobrazeni1,0,14,true));
levyPanel.add(getNapis("cislo A je",zobrazeni2,0,2,true));
levyPanel.add(getNapis("delitelne DELITEL",zobrazeni2,-5,14,true));
levyPanel.add(getNapis("cislo A neni",zobrazeni3,0,2,true));
```

Obrázek 42: Nastavení sekundárních nápisů v objektech

Dále následuje nastavení speciálních šipek v diagramu. Metody na vytvoření těchto šipek má každý objekt v sobě, tudíž šipka začíná u objektu, u kterého je vyvolána příslušná metoda na vytvoření šipky. Šipka končí u objektu, který je uveden v parametru metody, kde se dá v případě potřeby uvést i prodloužení šipky.

Tím je hotov konstruktor a dále následují dvě abstraktní metody z třídy „DiagramPanel“, které je potřeba nyní implementovat. První metoda se jmenuje „zablokovatObjekty“ (viz Obrázek 43) a je vykonána při každém označení libovolného objektu. A podle současného stavu proměnných jsou zablokovány nebo odblokovány určité objekty.

```
@Override
//Metoda pro zablokování objektu,
//který nesmí být označen za určitých podmínek.
void zablokovatObjekty(){
    if(promenneStranka.promenna[1]==0){
        zobrazeni3.nastavitNeoznacitelny();
        zobrazeni2.nastavitNeoznacitelny();
        podminka2.nastavitNeoznacitelny();
        vypocet.nastavitNeoznacitelny();
        zobrazeni4.nastavitOznacitelny();
    }else{
        if(promenneStranka.promenna[0]%promenneStranka.promenna[1] == 0){
            zobrazeni3.nastavitNeoznacitelny();
            zobrazeni4.nastavitNeoznacitelny();
            zobrazeni2.nastavitOznacitelny();
            podminka2.nastavitOznacitelny();
            vypocet.nastavitOznacitelny();
        }else{
            zobrazeni2.nastavitNeoznacitelny();
            zobrazeni4.nastavitNeoznacitelny();
            zobrazeni3.nastavitOznacitelny();
            podminka2.nastavitOznacitelny();
            vypocet.nastavitOznacitelny();
        }
    }
}
```

Obrázek 43: Nastavení označitelnosti objektů podle aktuálního stavu proměnných

Druhá metoda s názvem „vypocitat“ (viz Obrázek 44) se vykoná při každém označení objektu, který má nějaké proměnné přiřazovat hodnotu. V metodě je tedy proveden výpočet a příslušné proměnné přiřazena daná hodnota.

```
@Override
//Metoda pro vypočítání proměnné v programu.
void vypocitat(Objekt objektProVypocet) {
    if (objektProVypocet.equals(vypocet)){
        promenneStranka.promenna[2] = promenneStranka.promenna[0] % promenneStranka.promenna[1];
    }
}
```

Obrázek 44: Konkrétní implementace metody „vypočítat“

Třída „ZdrojovyKodStranka“

Veškerý algoritmus příkladu je definován ve třídě pro diagram a zde již stačí pouze vytvořit objekty představující řádky (viz Obrázek 45). Řádky jsou vytvořeny pomocí třídy „Text“, které jsou jako parametry zadány: samotný text, který má být zobrazen, pravdivostní hodnota určující zda má být řádek označitelný, pořadí v panelu a vzdálenost od levé strany panelu (počet tabulátorů).

```
//Vytvoření konkrétních řádků.
radek1 = new Text("program ZjisteniDelitelnosti;",false,1,0);
radek2 = new Text("var A, DELITEL, ZB: Integer;",false,2,0);
prazdnyRadek = new Text("  ",false,3,0);
radek3 = new Text("begin",true,4,0);
radek4 = new Text("Writeln ('Zadejte cislo a jeho delitele:');",true,5,1);
```

Obrázek 45: Vytvoření konkrétních řádků ve zdrojovém kódu

Dále se musí metodě pro automatické zezelenání textu předat identifikátory proměnných a metoda právě tyto identifikátory zvýrazní zeleně.

A jako poslední je potřeba vytvořit komentáře pomocí třídy „Komentar“, které stačí do parametrů zadat text představující komentář a řádek, za kterým má být komentář zobrazen.

Třída „PromenneStranka“

V této třídě stačí pouze vytvořit nápisy proměnných pomocí metody „pridatPromennou“, ve které je zadán jako parametr pouze statický text uvedený před hodnotou proměnné.

Jako poslední je třeba definovat abstraktní metodu „navazatNaRadek“ z rodičovské třídy. Tato metoda je volána při každé změně jakékoli proměnné. V této metodě je nastavena řádkům, které mají vypisovat nějakou proměnnou, aktuální hodnota této proměnné.

Závěr

V teoretické části bakalářské práce rozebírám současné přístupy k výuce základů programování. Konkrétně nejdříve porovnávám dva druhy paradigmat programování (strukturované a objektově orientované paradigma), která dnes rozdělují autory učebnic pro základy programování na dva tábory.

Po porovnání těchto dvou paradigmat se ubírám ještě hlouběji do dané problematiky a porovnávám již jednotlivé jazyky, které autoři považují za výukově vhodné pro začátečníky. V tomto případě se již autoři rozcházejí rovnou na několik skupin a každý má pro použití jím vybraného jazyka také jiné argumenty.

Všechna tato hlediska jsem zohlednil a vybral jsem sice již dnes lehce zastaralý, ale podle mého názoru stále pro tyto účely velice vhodný jazyk Pascal. Tento jazyk jsem vybral hlavně z důvodu jeho jednoduchosti a také proto, že má dobrý vliv na začínajícího programátora, což je důsledkem toho, že byl pro výuku programování přímo vytvořen. Nicméně za jeho silného konkurenta bych považoval dnes moderní jazyk Python, který by mohl v budoucnu Pascal dokonce úplně vytlačit ze škol pro jeho jednoduchost a široké využití.

Jako praktickou část bakalářské práce jsem vytvořil interaktivní výukový materiál v podobě programu pro výuku základů programování, který se nachází na přiloženém CD. Program je směřován na úplné začátečníky v programování a výuka v něm probíhá v jazyce Pascal.

Co se týče obsahu látky v programu, snažil jsem se do probírané látky zahrnout pouze ta témata, která považuju za opravdu základní pro programování a také ta, která nejsou výhradou jazyka Pascal. Vzhledem k tomu, že je program směřován na absolutní začátečníky, je důraz kladen spíše na příklady a v teorii jsou vysvětleny pouze nejdůležitější poznatky pro následující využití příkladu.

Program se dá využít pro výuku přímo ve školní hodině nebo jako dobrovolná pomůcka na domácí procvičení, případně ji může kdokoli využít i nezávisle na školní výuce, protože vysvětlení veškeré potřebné teorie je přímo jeho obsahem.

Tomuto tématu bych se v budoucnu ještě rád věnoval podrobněji, neboť se domnívám, že by program podobný tomu, který zde prezentuji, mohl být skutečně využit v praxi při vyučování základů programování a mohl by tak umožnit mnoha začátečníkům nahlédnout do světa programovaní jednoduchým způsobem. Bylo by zajímavé, avšak mnohem náročnější, rozšířit program tak, aby příklady v něm nemusely být pouze zadané programátorem, ale aby si samotný uživatel mohl svůj zdrojový kód vložit do programu a následně si ho krokovat.

Seznam použitých zdrojů

[1] Programovací paradigmata. *ARI* [online]. [cit. 2013-03-27]. Dostupné z: http://ari.wikidot.com/programovaci-paradigmata

[2] SATRAPA, Pavel. *Pascal pro zelenáče*. Vyd. 4. Praha: Neocortex, 2001, 253 s. ISBN 80-863-3003-6.

[3] VYSTAVĚL, Radek. *Moderní programování: učebnice pro začátečníky*. 3. vyd. Ondřejov: moderníProgramování, 2009, 195 s. ISBN 978-80-903951-6-9.

[4] DAŘENA, František. Objektově orientované programování. *Akela.mendelu.cz: studentský server* [online]. 2004 [cit. 2013-03-27]. Dostupné z: https://akela.mendelu.cz/~darena/Perl/objekty.html

[5] TIŠNOVSKÝ, Pavel. Programovací jazyky určené pro výuku programování (2). *Root.cz* [online]. 2010 [cit. 2013-03-27]. Dostupné z: http://www.root.cz/clanky/programovaci-jazyky-urcene-pro-vyuku-programovani-2/

[6] TIŠNOVSKÝ, Pavel. Scratch: plnohodnotný programovací jazyk nebo jen dětské puzzle? *Root.cz* [online]. 2011 [cit. 2013-03-12]. Dostupné z: http://www.root.cz/clanky/scratch-plnohodnotny-programovaci-jazyk-nebo-jen-detske-puzzle/

[7] KONEČNÝ, Jan a Vilém VYCHODIL. *Paradigmata programování* 1, díl A. [online]. Olomouc, 2008, s. 330 [cit. 2013-03-29]. Dostupné z: http://phoenix.inf.upol.cz/esf/ucebni/pp1a.pdf

[8] OBBAYI, S. R. Structured vs. Object-Oriented Programming: A Comparison. *BRIGHT HUB: The hub for Bright Minds* [online]. 2010 [cit. 2013-03-12]. Dostupné z: http://www.brighthub.com/internet/web-development/articles/82024.aspx?cid=parsely_rec

[9] SKOUPIL, David. *Úvod do paradigmat programování* [online]. Olomouc, 2007 s. 73 [cit. 2013-03-29]. Dostupné z: http://phoenix.inf.upol.cz/esf/ucebni/uvod_para.pdf

[10] VIRIUS, Miroslav. *Java pro zelenáče*. 2. upr. vyd. Praha: Neocortex, 2005, 268 s. ISBN 80-863-3017-6.

[11] Něco málo o C#. *Csharp.aspone.cz: Komplexní informační web o programování v .net* [online]. [cit. 2013-03-12]. Dostupné z: http://csharp.aspone.cz/

[12] PITNER, Tomáš. Výuka programování na základní a střední škole. *Fakulta informatiky Masarykovy univerzity* [online]. Dostupné z: http://www.fi.muni.cz/~tomp/semuc/text_pitner.html

[13] BOSÁK, Tomáš. Úvod do programovacieho jazyka Java. *Programujte.com* [online]. 2006 [cit. 2013-03-12]. Dostupné z: http://programujte.com/clanek/2006041804-uvod-do-programovacieho-jazyka-java/

[14] What Is C++? *WiseGEEK: clear answers for common questions* [online]. [cit. 2013-03-12]. Dostupné z: http://www.wisegeek.org/what-is-c.htm

[15] A brief description of C++. In: *C++ Programming language* [online]. 2011 [cit. 2013-03-12]. Dostupné z: http://cplusplus-developer.blogspot.cz/2011/12/brief-description-of-c.html

[16] GLASSBOROW, Francis. *Naučte se programovat!: podrobný průvodce programováním v C++*. 1. vyd. Překlad Karel Voráček. Praha: Grada, 2005, 400 s. ISBN 80-247-1243-1.

[17] Python. *AbcLinuxu* [online]. 2006 [cit. 2013-03-12]. Dostupné z: http://www.abclinuxu.cz/software/programovani/jazyky/python

[18] Python. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-03-12]. Dostupné z: http://cs.wikipedia.org/wiki/Python

[19] HEROUT, Pavel. *Učebnice jazyka Java*. 3., rozš. vyd. České Budějovice: Kopp, 2007, 381 s. ISBN 978-80-7232-323-4

[20] SGP Baltík 3. *SGP systems* [online]. [cit. 2013-03-12]. Dostupné z: http://www.sgpsys.cz/cz/Product_B3.asp

[21] Booleova logika. *IT slovník.cz* [online]. [cit. 2013-03-13]. Dostupné z: http://it-slovnik.cz/pojem/booleova-logika

[22] MCCULLEY, Mark. Focus on Swing. *JAVAWORLD: solutions for java developers* [online]. 1998 [cit. 2013-03-15]. Dostupné z: http://www.javaworld.com/javaworld/jw-07-1998/jw-07-swing-focus.html

[23] *Rámcový vzdělávací program pro základní vzdělávání* (se změnami k 1. 9. 2010). [online]. Praha: Výzkumný ústav pedagogický v Praze, 2007. 126 s. [cit. 29. 3. 2012]. Dostupné z WWW: <http://www.vuppraha.cz/wp-content/uploads/2009/12/RVPZV_2007-07.pdf>